JIDOSHALIGHT
Manual de Integración

Revisión 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 los derechos reservados.

Visite nuestro sitio web http://www.pumatronix.com

Envíe comentarios sobre este documento a suporte@pumatronix.com

La información contenida en este documento está sujeta a cambios sin previo aviso.

Pumatronix se reserva el derecho de modificar o mejorar este material sin obligación de notificar cambios o mejoras.

Pumatronix otorga permiso para descargar e imprimir este documento, siempre que la copia electrónica o física de este documento contenga el texto completo. Cualquier alteración de este contenido está estrictamente prohibida.

Historial de Cambios

Fecha	Revisión	Contenido actualizado
22/12/2022	1.0	Versión Inicial

Información General

Este documento tiene como objetivo orientar al desarrollador en la aplicación de la biblioteca de software JidoshaLight encargada del reconocimiento y lectura automática de matrículas de vehículos (LPR) basada en análisis de imágenes y aplicable en software compatible con la biblioteca. Este documento detalla las opciones de configuración del kit de desarrollo de software (SDK) y las API disponibles.

Según la versión de la biblioteca aplicada al software, es posible que algunos formatos de matrículas de vehículos no sean compatibles y que algunas funciones solo estén disponibles en las versiones más actuales.

Índice

1 Estructura del SDK de JidoshaLight 5
1.1 APIs Compatibles 5
2 JidoshaLight Linux 6
2.1 Arquitectura de software de JidoshaLight Linux 6
2.1.1 Restricciones JidoshaLight Linux 8
2.2 Instalación de JidoshaLight Linux 8
2.2.1 *Configuración de permisos de hardkey* 8
2.2.2 Configuración de variables de entorno 8
2.2.3 Configuración del archivo de preferencias 10
2.3 Aplicaciones de muestra 10
3 JidoshaLight Windows 12
3.1 Instalación de JidoshaLight Windows 12
3.1.1 Configuración de variables de entorno 12
3.1.2 Configuración del archivo de preferencias 13
3.2 Aplicaciones de muestra 13
4 JidoshaLight Linux/FPGA 14
4.1 Arquitectura de software de JidoshaLight Linux/FPGA 14
4.1.1 Restricciones JidoshaLight Linux/FPGA 15
4.2 Instalación de JidoshaLight Linux/FPGA 15
4.2.1 Configuración de licencia 15
4.2.2 Configuración de variables de entorno 15
4.2.3 Configuración del *kernel* Linux 16
4.3 Aplicaciones de muestra 17
5 JidoshaLight Android™ 17
5.1 Arquitectura de software JidoshaLight Android™ 17
5.1.1 Restricciones JidoshaLight Android™ 18
5.2 Instalación de JidoshaLight Android™ 19
5.2.1 Licencia 19
5.2.2 Permisos 19
5.3 Aplicación de Muestra 20
5.3.1 Package `br.gaussian.io` 20
5.3.2 Package `br.gaussian.jidoshalight` 20
5.3.3 Package `br.gaussian.jidoshalight.camera` 20
5.3.4 Package `br.gaussian.jidoshalight.sample` 20
6 APIs de usuário 24
6.1 Limitaciones conocidas 24
6.2 API JidoshaLight C/C++ 25

6.2.1 API JidoshaLight C/C++ (Local) ... 25
6.2.2 API JidoshaLight C/C++ (Remota Síncrona) ... 44
6.2.3 API JidoshaLight C/C++ (Remota Assíncrona) ... 45
6.2.4 API JidoshaLight C/C++ (Servidor) ... 51
6.3 API JidoshaLight Java ... 53
6.3.1 API JidoshaLight Java (Local) ... 53
6.3.2 API JidoshaLight Java (Remota Asíncrona) ... 65
6.3.3 API JidoshaLight Java (Servidor) ... 70
6.3.4 API JidoshaLight Java (IO/Mjpeg) ... 72
6.4 Guía de Migración - API 1 C/C++ JIDOSHA ... 74
7 APIs de usuario de JIDOSHA ... 75
7.1 jidoshaCore.h ... 76
7.2 API1 JIDOSHA C/C++ ... 79
7.2.1 Tipos ... 79
7.2.2 Métodos ... 80
7.3 API2 JIDOSHA C/C++ ... 81
7.3.1 Tipos ... 81
7.3.2 Métodos ... 82
7.3.3 API 2 - Configuración ... 86
7.3.4 API 2 - Configuración de perspectiva de imagen ... 91
7.4 API JIDOSHA C# / VB.NET ... 93
7.4.1 API 1 ... 93
7.4.2 Ejemplos API JIDOSHA C# / VB.NET ... 95
7.5 API JIDOSHA Delphi ... 97
7.5.1 API 1 ... 97
7.5.2 Ejemplo API JIDOSHA Delphi ... 98
7.6 API JIDOSHA Java ... 98
7.6.1 API 1 ... 98
7.6.2 Ejemplo API JIDOSHA Java ... 99
7.7 Builds especiales de API heredadas ... 99

1 Estructura del SDK de JidoshaLight

Todas las rutas utilizadas en este manual son relativas al directorio raíz del SDK JidoshaLight_TARGET_x.yz El SDK es el kit de desarrollo de software de JidoshaLight compuesto por:

- bibliotecas de reconocimiento de matrículas libjidoshaLight.so, libjidoshaLightRemote.so, libjidoshaLightJava.so;
- respectivas APIs de bibliotecas;
- por wrappers (bindings) para otros idiomas;
- por aplicaciones de muestra precompiladas;
- por el código fuente de estas aplicaciones;
- por un script de compilación básico;
- por el Manual de Integración;
- para una imagen de placa de matrícula de prueba

La estructura del paquete de datos SDK contiene:

Carpeta	Contenido
res	carpeta que contiene los manuales y una foto para usar como ejemplo al ejecutar el software
lib	carpeta que contiene las bibliotecas (.so o .dll)
includes	carpetas con headers para incluir en aplicaciones C
sample	carpeta que contiene los códigos de muestra para C *Nota: En la carpeta sample/bin hay mini programas para probar el funcionamiento.
wrappers	carpeta que contiene bindings para otros tipos de idiomas
jidoshapc	carpeta que contiene bindings para la antigua interfaz de Jidosha

1.1 APIs Compatibles

La API principal de Jidosha (interfaz de programación de aplicaciones) es la API JidoshaLight C/C++ y se puede encontrar dentro de las carpetas include y lib. Los wrappers (bindings) para otros idiomas se proporcionan junto con el SDK y se encuentran dentro de la carpeta de 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 la integración con otros idiomas aún no admitidos, comuníquese con el Soporte técnico.

El SDK también proporciona un conjunto de API heredadas dentro de la carpeta legacy. Estas APIs no reciben funciones nuevas y solo existen para admitir aplicaciones heredadas desarrolladas a partir de Jidosha (versión 1.7.0 o anterior). Internamente, esta API utiliza la API estándar JidoshaLight C/C++ y, por lo tanto, genera los mismos resultados de reconocimiento.

Atención a las APIs que NO se recomiendan para nuevos diseños:

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

2 JidoshaLight Linux

La biblioteca de software JidoshaLight Linux se creó para funcionar junto con la *hardkey* (clave de seguridad) que viene con la biblioteca. En otras palabras, para que la biblioteca funcione correctamente, esa *hardkey* debe estar conectada al USB del entorno en el que se utilizará la biblioteca. Existen dos versiones *hardkey*, una para uso general y la versión de demostración, con fecha de caducidad. Cuando vence la fecha de caducidad, la biblioteca comienza automáticamente a devolver las placas vacías. Si su *hardkey* de demostración caduca y desea comprar una licencia o extender el período de demostración, comuníquese con Pumatronix.

Compruebe los requisitos previos de instalación explicados en el Manual del Producto.

2.1 Arquitectura de software de JidoshaLight Linux

Las llamadas a la API de la biblioteca se pueden realizar de forma local o remota a través de una red IP.

Las llamadas locales se ejecutan en el mismo thread donde se realizó la llamada. Para licencias con más de 1 thread habilitado o para casos en los que el thread principal no se puede bloquear mientras se procesa la imagen, se deben crear nuevos thread para su procesamiento.

Las llamadas remotas pueden ser síncronas o asíncronas. En ambos casos las llamadas se realizan localmente y las imágenes se procesan remotamente en un servidor. La licencia de uso sólo es necesaria en el servidor que ejecuta el algoritmo, no siendo necesaria para el uso de la biblioteca remota.

Las llamadas sincrónicas se bloquean y devuelven el resultado del procesamiento al final de la llamada.

En el caso de la interfaz asíncrona, la llamada regresa inmediatamente y el resultado del procesamiento se devuelve a través de una *callback* del usuario.

La siguiente figura presenta un diagrama con la arquitectura sugerida para una aplicación Linux que utiliza la biblioteca JidoshaLight con llamadas locales, ya sea de un solo thread o multithread. Para que la aplicación funcione correctamente, la biblioteca *'libjidoshaLight.so'* debe estar vinculada a la aplicación y la hardkey debe estar conectada a la máquina. Luego, para el caso de un solo thread, simplemente llame a las funciones API. En cuanto al caso multithread, la aplicación debe crear los threads de procesamiento necesarios y, a partir de estos, realizar llamadas a las funciones API de la biblioteca JidoshaLight.

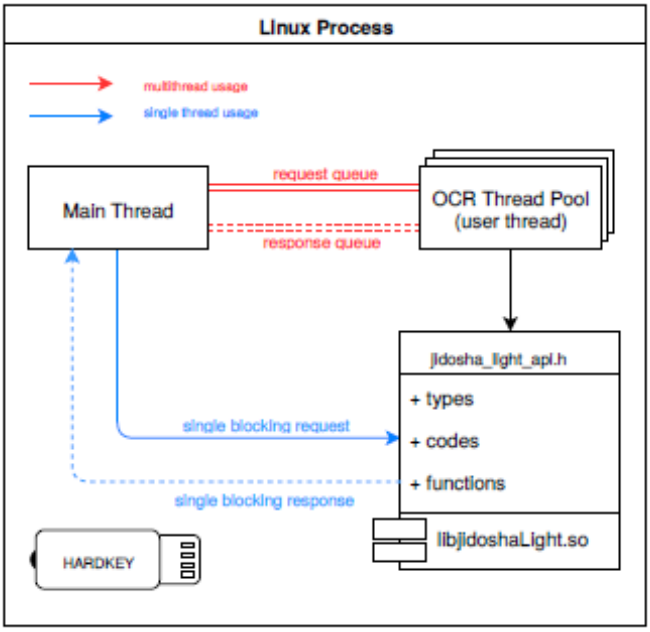


Figura 1 – Diagrama con la arquitectura sugerida para una aplicación Linux

La siguiente figura muestra la arquitectura sugerida para usar la biblioteca con llamadas remotas. Para que la aplicación funcione correctamente, la biblioteca *'libjidoshaLightRemote.so'* debe estar vinculada a la aplicación cliente. La biblioteca *'libjidoshaLight.so'* debe estar vinculada a la aplicación del servidor y la hardkey debe estar conectada a la máquina. Las aplicaciones cliente y servidor deben estar interconectadas a través de una red TCP/IPv4 real o virtual (loopback, por ejemplo). Aunque no se ilustra en la figura, al igual que para las llamadas locales, la aplicación cliente puede tener varios threads y el servidor puede limitar la cantidad de sesiones activas simultáneas según la licencia.

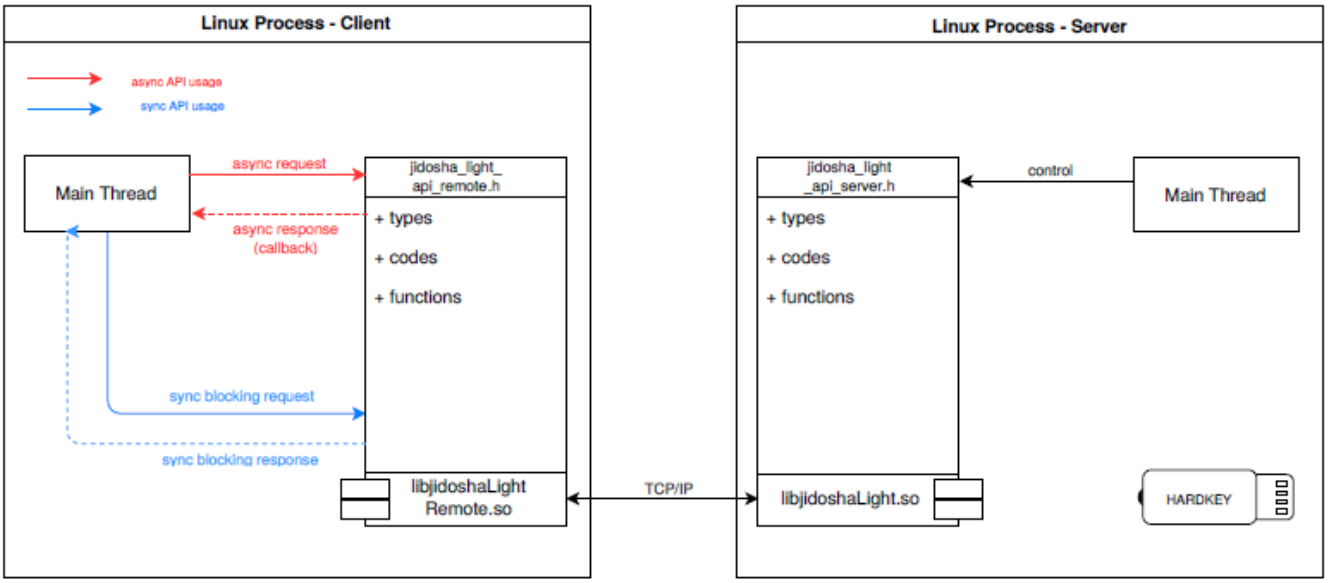


Figura 2 – Diagrama con la arquitectura sugerida para el uso de la biblioteca con llamadas remotas

2.1.1 Restricciones JidoshaLight Linux

La biblioteca admite aplicaciones multiproceso y multiproceso, con el número máximo de threads para todos los procesos limitado por la licencia adquirida.

La biblioteca no admite el proceso *fork*.

2.2 Instalación de JidoshaLight Linux

2.2.1 *Configuración de permisos de hardkey*

Para que la hardkey USB funcione correctamente, se deben cambiar los permisos de acceso de **udev**. Para su comodidad, el script *'res/scripts/install_udev.sh'* está incluido en el SDK de Jidosha, ejecutándolo con el argumento '*-i*', los permisos se instalarán automáticamente, también es posible ejecutar el script sin argumento para ver las opciones. Si prefiere instalar manualmente, siga estos pasos:

Agregue la siguiente línea:

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

al final del archivo correspondiente a su distribución de 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
```

Para Debian, agregue las líneas:

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

Y al final del archivo:

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

Para obtener instrucciones sobre cómo habilitar la hardkey en otras distribuciones de Linux, comuníquese con Pumatronix Equipamentos Eletrônicos.

2.2.2 Configuración de variables de entorno

Antes de ejecutar las aplicaciones de prueba proporcionadas con el SDK, o cualquier otra aplicación que utilice la biblioteca JidoshaLight Linux, es necesario configurar algunas variables de entorno para el correcto funcionamiento de la biblioteca.

Inicialmente, es necesario agregar el directorio que contiene las bibliotecas a la ruta de búsqueda del sistema, de la siguiente manera:

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

2.2.2.1 Sistema de log y auditoría

Atención: desde la versión 3.3.0 en adelante, el sistema de log de la biblioteca está DESHABILITADO por defecto.

El SDK de JidoshaLight tiene un sistema de log que se puede usar para auditar el comportamiento de la biblioteca en el campo. Para habilitar algunos mensajes de depuración preconfigurados, simplemente exporte la variable de entorno JL_LOGCFG con el valor "default".

```
$ export JL_LOGCFG=default
```

El sistema de registro también permite habilitar otros mensajes de depuración y redirigir el contenido de estos mensajes a uno o más archivos. Esta funcionalidad se configura a través de un archivo de configuración cuya estructura se especifica a continuación.

La lectura del archivo de configuración ocurre solo 1 vez durante la carga de la biblioteca y tiene el siguiente orden de búsqueda:
1. ruta absoluta indicada por la variable de entorno *JL_LOGCFG* (si está definida)
2. archivo *jlog.conf* en el directorio actual [./]

Estructura del archivo de configuración del 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
```

El archivo de configuración de registro anterior hará que la biblioteca genere todos los mensajes habilitados, tanto para el archivo con la ruta relativa *'log.txt'* como para la salida estándar (stdout). Si desea inhibir uno o más tipos de mensajes, simplemente comente la línea con '#' o elimínela.

Para inhibir mensajes a stdout y escribir solo en el archivo log.txt, siga el ejemplo a continuación para cada tipo de mensaje deseado:

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

2.2.3 Configuración del archivo de preferencias

La biblioteca permite el uso opcional de un archivo de preferencias. A través de este archivo es posible configurar los campos de la *'struct JidoshaLightConfig'*, sobrescribiendo los campos de esta *'struct'* pasados por la API. El formato es json, de la siguiente manera:

```
{
"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
            }
    }
}
```

De forma predeterminada, la biblioteca busca el archivo jl_anpr_preferences.json en el directorio de trabajo. Si el archivo existe, se cargará; en caso contrario se buscará en la ruta indicada por la variable de entorno JL_ANPR_PREFS. Si la variable no existe, no se carga ningún archivo de preferencias. Si existe y la ruta indicada es un archivo válido, se carga.

Si se cargó un archivo de preferencias, las preferencias presentes en él solo se aplicarán si el campo *'ENABLE_CONFIG_OVERRIDE'* es *'true'*. Los campos de *'struct JidoshaLightConfig'* que faltan en el archivo de preferencias recibirán el valor default, según lo define la biblioteca.

Dado que el archivo de preferencias se carga al inicio de la biblioteca (generalmente al comienzo del proceso que lo usa), las modificaciones al archivo solo surtirán efecto cuando se vuelva a cargar la biblioteca. Este comportamiento puede cambiar en versiones futuras.

2.3 Aplicaciones de muestra

El SDK incluye algunas aplicaciones de muestra con código fuente incluido:

- *JidoshaLightSample*: Ejemplo de procesamiento local
- *JidoshaLightSampleClient*: Ejemplo de aplicación de cliente con procesamiento remoto asíncrono

- *JidoshaLightSampleServer*: Ejemplo de aplicación de servidor
- *JidoshaLightSampleAsync*: Ejemplo de procesamiento local asíncrono de multithreads
- *JidoshaLightSampleMulti*: Ejemplo de reconocimiento de varias matrículas en una misma imagen

Si desea volver a compilar los ejemplos, use el script make_samples.sh:

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

Después de configurar la *hardkey* las variables de entorno y conectar la *hardkey*, será posible ejecutar los ejemplos.

Para ejecutar *JidoshaLightSample*, ejecute el programa de ejemplo con la imagen de la placa de referencia desde la terminal:

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

La aplicación debe informar la versión de la biblioteca, así como el resultado del reconocimiento de imágenes.

```
-- 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 ejecutar los ejemplos *JidoshaLightSampleServer* y *JidoshaLightSampleClient*, ejecute el programa del servidor desde una terminal:

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

La aplicación debe informar que se inició utilizando el puerto TCP 51000 y se encontró la *hardkey*.

```
[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
```

Luego, en otra terminal, ejecute el programa cliente. El cliente debe informar la versión de la biblioteca, así como el resultado/estadísticas del reconocimiento de imágenes:

```
$ ./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 %)
```

Volviendo a la terminal del servidor, busque mensajes de registro adicionales que informen datos de licencia y eventos de conexión:

```
[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

La biblioteca de software *JidoshaLight Linux* se creó para funcionar junto con la *hardkey* (clave de seguridad) que viene con la biblioteca. En otras palabras, para que la biblioteca funcione correctamente, esa *hardkey* debe estar conectada al USB del entorno en el que se utilizará la biblioteca. Existen dos versiones *hardkey*, una para uso general y la versión de demostración, con fecha de caducidad. Cuando vence la fecha de caducidad, la biblioteca comienza automáticamente a devolver las placas vacías. Si la *hardkey* de demostración caduca, puede comprar una licencia o extender el período de demostración comunicándose con Pumatronix.

Compruebe los requisitos previos de instalación explicados en el Manual del Producto.

3.1 Instalación de JidoshaLight Windows

Para la instalación, solo es necesario conectar la *hardkey* a una máquina con Windows que ejecutará el software; luego, Windows debería instalar un controlador automáticamente la primera vez. Para comprobar si la instalación se ha realizado correctamente, puede ejecutar las aplicaciones de muestra, detalladas en la Configuración del archivo de preferencias.

3.1.1 Configuración de variables de entorno

Antes de ejecutar las aplicaciones de prueba proporcionadas con el SDK, o cualquier otra aplicación que utilice la biblioteca JidoshaLight Windows, es necesario configurar algunas variables de entorno para el correcto funcionamiento de la biblioteca.

1. Inicialmente, es necesario agregar el directorio que contiene las bibliotecas a la ruta de búsqueda del sistema, para eso, acceda a la carpeta SDK y escriba el comando:

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

NOTA: El comando anterior solo cambiará la PATH para la sesión de terminal abierta, si necesita configurar el entorno, debe acceder al panel de control > sistema > cambiar configuraciones > propiedades del sistema > avanzado > variables de entorno y allí cambiar el valor de sistema o variable de usuario llamada Path.

3.1.1.1 Sistema de log y auditoría

Ver Sistema de log y auditoría utilizado en JidoshaLight Linux.

3.1.2 Configuración del archivo de preferencias

Ver Configuración del archivo de preferencias usado en JidoshaLight Linux.

3.2 Aplicaciones de muestra

El SDK incluye algunas aplicaciones de muestra con código fuente incluido:

- *JidoshaLightSample:* Ejemplo de procesamiento local
- *JidoshaLightSampleClient:* Ejemplo de aplicación de cliente con procesamiento remoto asíncrono
- *JidoshaLightSampleServer:* Ejemplo de aplicación de servidor
- *JidoshaLightSampleAsync:* Ejemplo de procesamiento local asíncrono de multithreads
- *JidoshaLightSampleMulti:* Ejemplo de reconocimiento de varias matrículas en una misma imagen
- *JidoshaLightSampleServerService:* Ejemplo de servidor como servicio de Windows

Para ejecutar *JidoshaLightSample*, desde la carpeta SDK, ejecute el siguiente comando y debería obtener un resultado similar:

```
>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

La biblioteca de software *JidoshaLight Linux* con aceleración FPGA se licencia a partir de un archivo de licencia adjunto al hardware, sin necesidad de usar una *hardkey* (llave de seguridad). Esta biblioteca admite aceleración de hardware basada en FPGA Xilinx de la familia **Zynq-7000**. Por defecto tiene soporte para el dispositivo XC7Z020-CLG400, pudiendo adaptarse para dispositivos de mayor capacidad.

Compruebe los requisitos previos de instalación explicados en el Manual del Producto.

4.1 Arquitectura de software de JidoshaLight Linux/FPGA

La arquitectura del software es similar a la versión de Linux no acelerada descrita en JidoshaLight Linux.

Las principales diferencias están en las interfaces de programación adicionales del dispositivo y en el área reservada de memoria compartida (*shared memory*). Los detalles de configuración e instalación son específicos y se describen en Instalación.

Las siguientes figuras ilustran la arquitectura sugerida para las API locales y remotas.

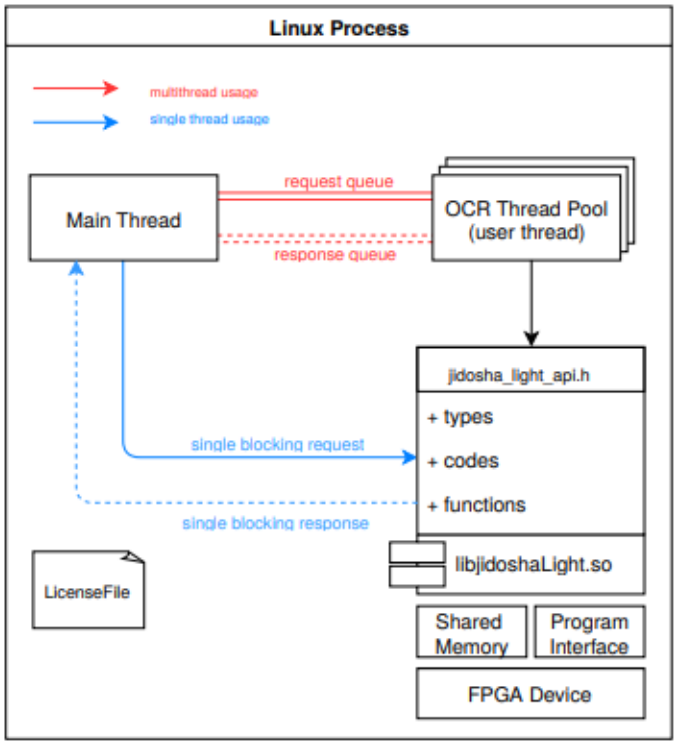


Figura 3 – Diagrama con casos de uso de API locales

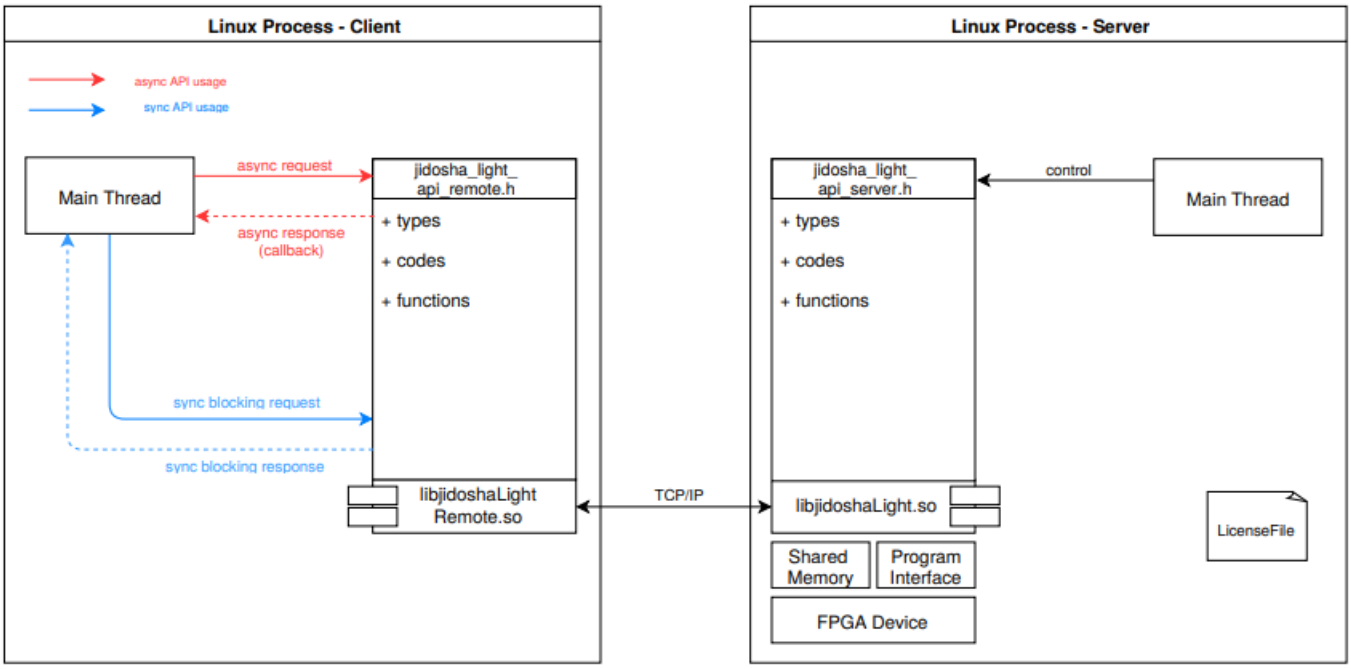


Figura 4 – Diagrama con casos de uso de API remota

4.1.1 Restricciones JidoshaLight Linux/FPGA

La biblioteca acelerada por FPGA admite aplicaciones de multithread, con el número máximo de threads limitado por la licencia adquirida. No hay soporte para aplicaciones multiproceso.

4.2 Instalación de JidoshaLight Linux/FPGA

4.2.1 Configuración de licencia

La biblioteca se licencia a través de un archivo vinculado al dispositivo utilizado.

Para obtener su identificador de hardware, ejecute la aplicación *JidoshaLightDna* desde el terminal del dispositivo correctamente configurado, como se describe en Configuración de variables de entorno.

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

IMPORTANTE: No se permite el uso simultáneo de esta aplicación con cualquier otra que use la biblioteca y puede causar bloqueos.

4.2.2 Configuración de variables de entorno

Antes de ejecutar las aplicaciones de prueba proporcionadas con el SDK, o cualquier otra aplicación que utilice la biblioteca JidoshaLight Linux, es necesario configurar algunas variables de entorno para el correcto funcionamiento de la biblioteca.

Cuando se utiliza la versión acelerada por FPGA, además de las variables descritas en Configuración de variables de entorno, se requieren los ajustes que se describen a continuación:

- JL_MPOOL_BASE: Dirección base de memoria destinada a la comunicación entre la biblioteca y la FPGA. Si no se define, el valor predeterminado es 0x3A000000. Ej.:

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

- JL_MPOOL_BUFFERNUM: número de *buffer* de memoria necesarios para la ejecución de la biblioteca. Si no se define, el valor predeterminado es 32 búferes, siendo el valor mínimo requerido 12 *buffer* por *thread* que usa la biblioteca simultáneamente. Ej.:

```
$ export JL_MPOOL_BUFFERNUM=32
```

- JL_MPOOL_BUFFERSIZE: Tamaño de cada memoria *buffer*, siendo múltiplo de 4096 *bytes*. Si no se define, el valor predeterminado es 2097152 *bytes* (2 MB). Este es el valor necesario para procesar imágenes de hasta 800x600 píxeles. Este valor debe ser superior a 4 veces la resolución de la imagen. Ej.:

```
$ export JL_MPOOL_BUFFERSIZE=2097152
```

- JL_LICENSE_FILE: Ruta al archivo de licencia. El archivo de licencia está vinculado al dispositivo utilizado. Para el identificador, consulte Configuración de licencia. Ej.:

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

4.2.3 Configuración del *kernel* Linux

Para la comunicación entre la biblioteca y el dispositivo FPGA, se requieren dos interfaces, una dedicada a la configuración de FPGA y una memoria compartida para el intercambio de datos.

Xilinx proporciona la interfaz de configuración a través de un *char device* (/dev/xdevcfg) y actualmente no forma parte del *kernel* estándar de Linux. El código fuente y las instrucciones de instalación se pueden consultar en la página Wiki de Xilinx.

La memoria compartida debe estar visible en /dev/mem y reservarse para uso exclusivo de la biblioteca y no puede ser utilizada por el *kernel* de Linux.

Para hacerlo, es necesario limitar la cantidad de memoria utilizada por el *kernel* al iniciarlo.

A continuación, se muestra un ejemplo para reservar los últimos 96 MB de memoria en un dispositivo con 1 GB de RAM. En u-boot, configure:

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

Para que el dispositivo */dev/mem* esté disponible, use la opción *CONFIG_DEVMEM=y* en kconfig en el proceso de compilación del *kernel*.

Agregue también al *device tree* de Linux (DTS) las siguientes configuraciones:

```
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 Aplicaciones de muestra

Ver Aplicaciones de muestra utilizadas para JidoshaLight Linux.

5 JidoshaLight Android™

La biblioteca de software *JidoshaLight Android™* se creó para funcionar junto con el archivo de licencia que se debe generar después de que el usuario instala la aplicación. El archivo de licencia se genera por instalación y está vinculado al hardware del dispositivo, requiriendo una nueva licencia en caso de reinstalar la aplicación o modificar el hardware del dispositivo, incluida la tarjeta SIM del dispositivo. El reemplazo de la batería no requiere una nueva licencia. Para licencias temporales, liberadas por tiempo limitado, la fecha y hora del equipo debe estar sincronizada con la red celular.

La biblioteca admite aplicaciones de multithread, con el número **máximo de threads** y el **tiempo de procesamiento mínimo** limitado por la licencia adquirida. Para el uso de la API del servidor, el número **máximo de conexiones simultáneas** que acepta también está limitado por la licencia.

Se accede a las funcionalidades de la biblioteca de *JidoshaLight Android* a través de la API de Java. Esta versión es compatible con procesadores ARM™ (armv7-a) con Android™ 4.4 o superior para usar la biblioteca (shared libraries y clases básicas de Java) y Android™ 8 o superior para instalar la aplicación de demostración.

5.1 Arquitectura de software JidoshaLight Android™

La forma más recomendada de trabajar con la biblioteca JidoshaLight en la plataforma Android es a través de la topología *cliente asíncrona* y *servidor*. Esta topología permite optimizar el flujo del proceso de reconocimiento de matrículas, ya que todo el procesamiento y asignación de memoria se realiza en código nativo. Esta topología aún permite procesar las imágenes externamente sin cambiar la aplicación. La aplicación de demostración que viene con el SDK implementa esta topología.

Por lo general, se obtiene una mejor experiencia de usuario en el modo *freeflow*. En el modo *freeflow*, a diferencia de *point and shoot*, el proceso de reconocimiento de matrículas se realiza en todas las imágenes enviadas por la cámara, sin necesidad de intervención del usuario (disparo). Por lo tanto, tan pronto como se activa la cámara, comienza el procesamiento y se generan de forma asincrónica *callbacks* con los resultados del reconocimiento. La topología **cliente asíncrono y servidor** puede funcionar en modo *freeflow* sin ningún cambio significativo en la implementación de la aplicación.

Puntos **positivos** del cliente asíncrono de freeflow:

1. Simplificación del código de la aplicación, lo que facilita la integración de la biblioteca
2. Mejor experiencia de usuario (reconocimientos más rápidos)
3. Gestión automática de recursos (colas, threads, red)
4. Mayor desacoplamiento entre adquisición de imágenes (cámara), procesamiento (LPR) y salida (UI y DB)
5. Capacidad de procesamiento local o remoto sin modificación del código fuente

Puntos **negativos** del cliente asíncrono de freeflow:

1. Las callbacks ocurren en un thread separado del thread de la interfaz de usuario, lo que requiere sincronización (runOnUiThread)
2. Las callbacks se emiten secuencialmente y no se pueden bloquear (el código de callback debe ser ligero y rápido)
3. El manejo de errores asincrónicos suele ser más complejo

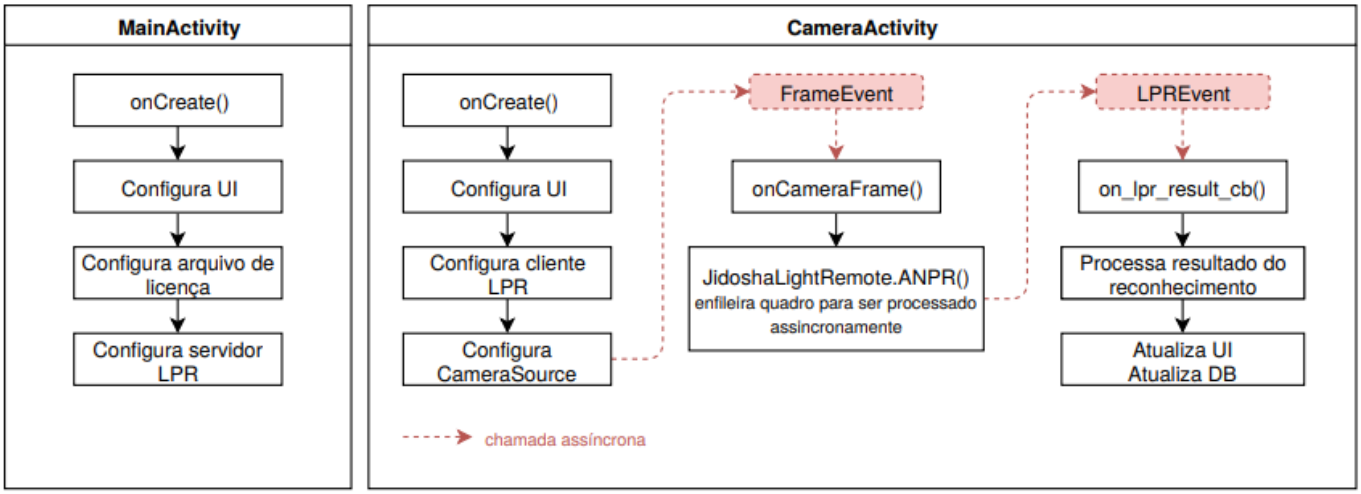


Figura 5 - Ejemplo de flujo para una aplicación cliente asíncrono de freeflow: MainActivity configura la licencia de la biblioteca e inicia el servidor de procesamiento, CameraActivity configura el cliente del lector de matrículas, la cámara (CameraSource) y maneja los eventos

5.1.1 Restricciones JidoshaLight Android™

ATENCIÓN: Las siguientes restricciones de memoria no se aplican a la memoria asignada de forma nativa (dentro de la *shared library*).
Para aplicaciones de alto rendimiento, se recomienda utilizar la API de cliente asíncrono.

El sistema operativo Android™ tiene restricciones estrictas sobre el uso de RAM por parte de las aplicaciones. La cantidad máxima de memoria que una aplicación puede asignar en el *heap* varía entre dispositivos, pero es de alrededor de 24 MB a 36 MB. Si una aplicación intenta asignar más memoria de la que se le permite, se genera una excepción *'OutOfMemoryError'* y el sistema operativo finaliza la aplicación.

Dado que la resolución de las cámaras de los *smartphones* y las tabletas está aumentando, el desarrollador debe prestar atención al tamaño de las imágenes que pretende reconocer para mitigar la posibilidad de una excepción del tipo *'OutOfMemoryError'*. Por ejemplo, una imagen de 8MP en formato Bitmap ARGB8888 ocupa 24MB, lo que sería suficiente para superar el límite de memoria en muchos dispositivos.

Como JidoshaLight necesita que **los caracteres de las matrículas tengan un máximo de 30 píxeles de alto**, una imagen con una resolución de **1280x720** es suficiente para el reconocimiento de matrículas. Si desea mostrar una imagen de alta resolución al usuario, puede adquirir y almacenar la imagen en alta resolución y, para su procesamiento, utilizar los métodos de decodificación con reducción de resolución admitidos por la clase *'android.graphics.Bitmap'* de Android™. En este caso, se debe tener en cuenta la reducción de caracteres en el proceso de reducción del tamaño de la imagen, asegurando el tamaño de 15 a 30 píxeles en la imagen reducida.

Otra especificidad importante del sistema operativo Android™ está relacionada con el bloqueo del thread de la interfaz gráfica. De forma predeterminada, el thread de la interfaz gráfica es el único creado por la aplicación y todo el procesamiento se realiza en él. Para que la interfaz gráfica siga respondiendo a las acciones del usuario, no debe bloquearse durante más de unos pocos milisegundos. Si esto ocurre, el sistema operativo emitirá una alerta al usuario informando que la aplicación ha dejado de responder o simplemente detendrá la aplicación. Para obtener más información sobre la gestión de memoria en Android:

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

5.2 Instalación de JidoshaLight Android™

ATENCIÓN: Todas las clases de Java que tienen métodos marcados como native no pueden cambiar su *package*.
Todas las demás clases se pueden mover libremente.

El SDK de desarrollo de biblioteca de lectura de matrículas de vehículos *JidoshaLight* para Android viene con la API y bibliotecas compartidas en lenguaje C nativo (compartidas, a las que se puede acceder mediante cualquier código JNI), wrappers y bibliotecas para la interfaz Java y una aplicación de demostración descrita en Aplicación de Muestra.

5.2.1 Licencia

Se requiere un archivo de licencia válido para que la biblioteca *JidoshaLight* funcione en sistemas Android™. El licenciamiento se realiza por dispositivo y por tiempo, requiriendo una nueva licencia si el equipo tiene sus características de hardware modificadas o el plazo de la licencia ha vencido.

A API `JidoshaLight.setLicenseFromData()` debe usarse para pasar el contenido del archivo de licencia a la biblioteca y debe hacerse **antes** de cualquier otra llamada a otras funciones de la API. La clase `JidoshaLightAndroidHelper.java` también tiene algunas funciones de utilidad que ayudan en el proceso de carga de licencias.

El procedimiento de solicitud de licencia está completamente automatizado por la función `JidoshaLight.getLicenseFromServer`, requiriendo solo que el usuario registre el *Device ID* del dispositivo con Pumatronix. La clase `LicenseManagerFragment.java` de la aplicación de ejemplo muestra cómo solicitar el *Device ID* del dispositivo y cómo solicitar una licencia del servidor.

Para obtener más información acerca de las licencias de dispositivos, comuníquese con el Soporte Técnico.

5.2.2 Permisos

Los siguientes permisos son necesarios para que la biblioteca funcione y deben incluirse en el *'AndroidManifest.xml'* de la aplicación:

```
<!-- 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 Aplicación de Muestra

La aplicación de muestra que viene con el SDK está diseñada para usarse con Android™ Studio 4 o superior. Muestra cómo usar la biblioteca para reconocer matrículas de una transmisión de video proveniente de la cámara trasera del teléfono o una cámara MJPEG externa. También trae un ejemplo de implementación para la pantalla de configuración de parámetros de la biblioteca, actividad de la cámara con cuadrícula y soporte de zoom, lista de reconocimientos, detalles de reconocimiento, sistema de licencias e integración de la base de datos para buscar información relacionada con la placa de matrícula detectada.

5.3.1 Package `br.gaussian.io`

- *Mjpeg.java*: Classe wrapper sobre la API nativa del decoder MJPEG. Consulte la classe `MjpegCamera.java` para una implementación de nivel superior.

5.3.2 Package `br.gaussian.jidoshalight`

- *JidoshaLight.java*: Classe que contiene funciones API locales (reconocimiento de placas y licencias) y códigos de retorno de función
- *JidoshaLightRemote.java*: Classe que contiene funciones de API remota asíncrona
- *JidoshaLightServer.java*: Classe que contiene las funciones de la API del servidor

5.3.3 Package `br.gaussian.jidoshalight.camera`

- *BaseCameraSource.java*: Clase base para todas las implementaciones de cámara
- *BackCamera.java*: Implementación para la cámara trasera del smartphone
- *MjpegCamera.java*: Implementación para una cámara MJPEG externa
- *CameraFrame.java*: Clase que almacena un cuadro de cámara
- *CameraView.java*: View capaz de mostrar un flujo de imágenes desde una BaseCameraSource; proporciona soporte de cuadrícula, superposición de placas, zoom de pinza y selección de ROI

5.3.4 Package `br.gaussian.jidoshalight.sample`

5.3.4.1 Common

- *common/JidoshaLightAndroidHelper.java*: Clase auxiliar que contiene métodos de soporte para leer y escribir el archivo de licencia, así como otros métodos de utilidad.
- *common/JidoshaLightServerHelper.java*: Clase auxiliar que contiene métodos de soporte para iniciar el servidor LPR local.

5.3.4.2 Activities

- *MainActivity*: Activity principal de la aplicación, muestra cómo configurar el archivo de licencia e iniciar el servidor local de lectura de matrículas.

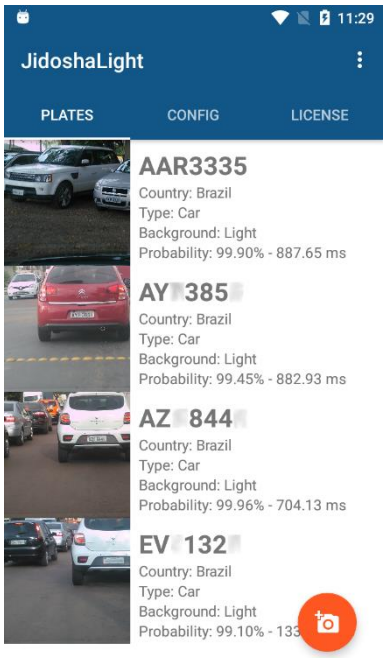


Figura 6 – MainActivity con caracteres ofuscados por razones de privacidad

- *DetailActivity*: Activity desencadenada al seleccionar un elemento de la lista de reconocimiento. Expande la información para un reconocimiento dado.

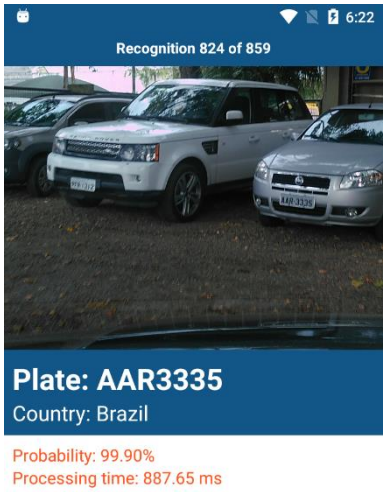


Figura 7 - DetailActivity

- *CameraActivity*: Ejemplifica el proceso de configuración y captura de imágenes de la cámara, permitiendo:

1. Crea una instancia de una cámara desde la configuración;
2. Configurar un cliente de procesamiento de placas;
3. Configurar el zoom óptico a través del movimiento de pinza;
4. Seleccione la región de interés (ROI) al tacto;
5. Habilitar/deshabilitar el procesamiento.

Para un mejor rendimiento de reconocimiento, el enfoque y el zoom de la cámara deben permitir capturar imágenes con buena nitidez y tamaño. La altura de la placa debe estar entre 30 y 50 píxeles. Las guías están destinadas a ayudar a enmarcar la placa, asegurando el tamaño y la orientación correctos de la placa en el momento de la captura. La placa debe tener aproximadamente el tamaño de un rectángulo de cuadrícula.

Debido a que la cámara del *smartphone* o *tablet* se mueve constantemente, es ideal, cuando esté disponible, activar las funciones de estabilización de video y enfoque automático del dispositivo. Según la aplicación, se recomienda exportar el enfoque manual y los ajustes de exposición para obtener los mejores resultados.

Figura 8 – CameraActivity: La altura ideal de la placa para el reconocimiento debe ser la misma que la de un rectángulo de cuadrícula (no es necesario que la placa esté alineada con la cuadrícula)

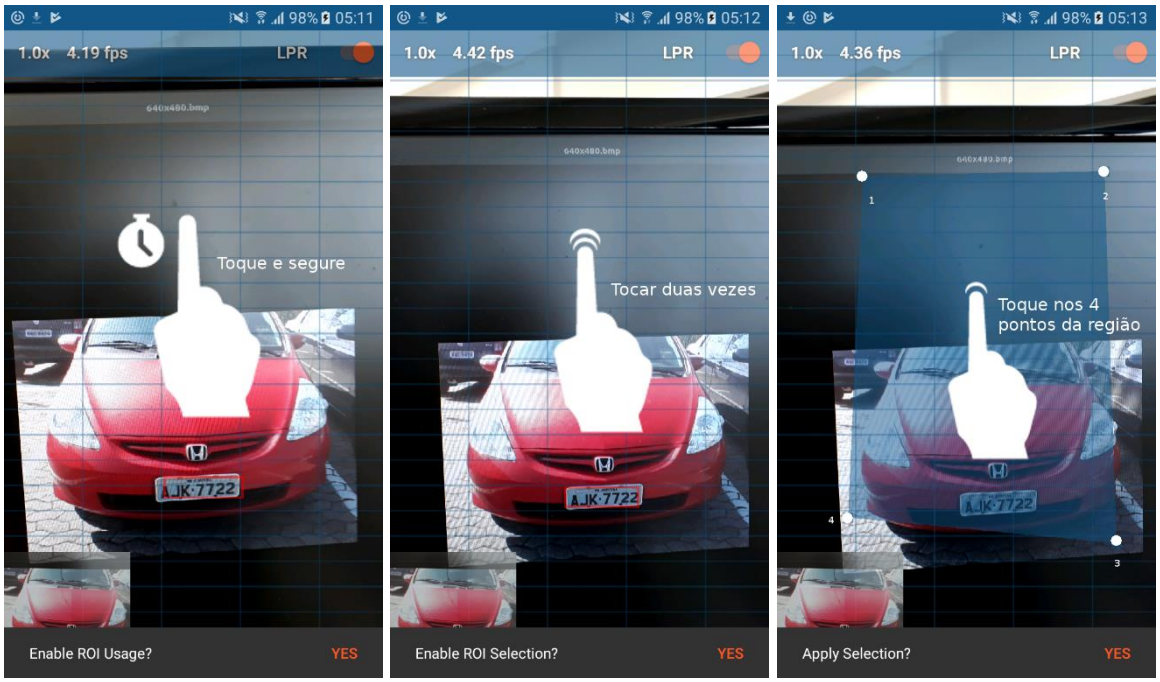
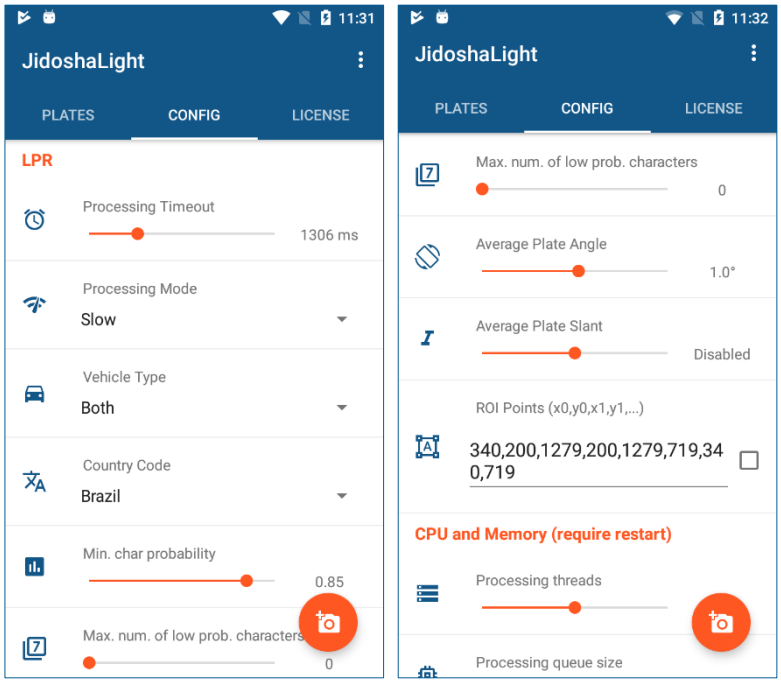


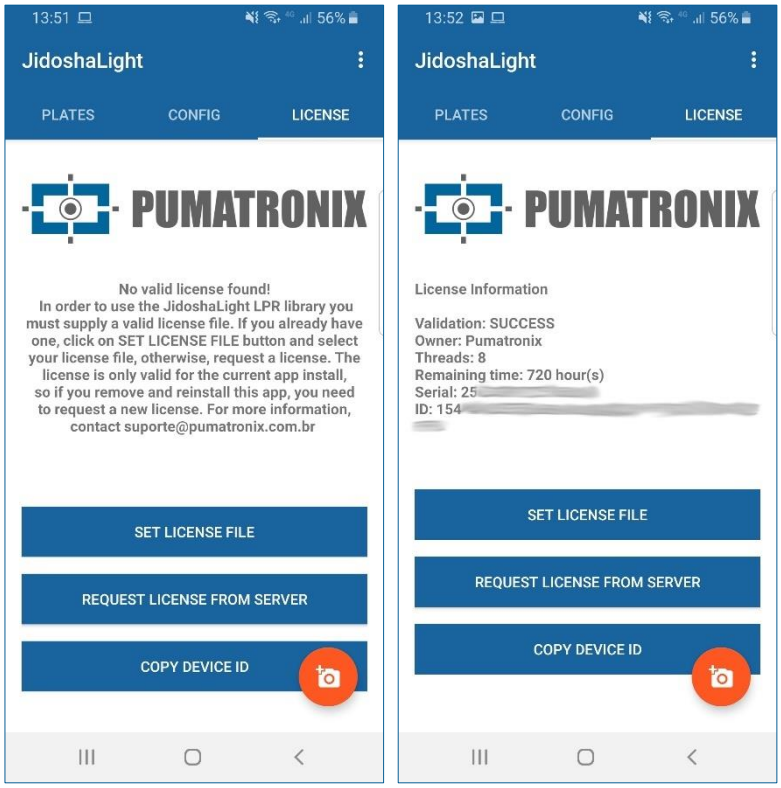
Figura 9 - Selección de la región ROI: 1) Toque y mantenga presionada la pantalla para habilitar el uso del ROI, 2) Toque dos veces para comenzar a seleccionar los puntos ROI, 3) Toque los cuatro puntos que delimitan la región (en sentido contrario).

5.3.4.3 Fragments

- *ConfigurationFragment*: Fragment utilizado para para mostrar y configurar los parámetros de la biblioteca JidoshaLight. Los parámetros configurables son los mismos disponibles en la API Java/C



- *LicenseManagerFragment*: Fragment utilizado para implementar el sistema de licencias. Muestra cómo leer el **Device ID** y cómo solicitar un archivo de licencia del servidor



Cambiar algunas configuraciones y el archivo de licencia requiere que se reinicie la aplicación.

6 APIs de usuário

La biblioteca *JidoshaLight* exporta 4 API diferentes para el reconocimiento automático de matrículas:

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

La API Remota Síncrona quedará obsoleto en el futuro y no se debe usar en proyectos nuevos. Además de las API de reconocimiento, la biblioteca proporciona algunas funciones de utilidad, como un receptor de video en formato MJPEG y un lector de licencias. Estas APIs complementarias están parcialmente documentadas en este manual. Para obtener más información sobre el uso, consulte los headers en la carpeta `include/gaussian/common.`

De forma predeterminada, los lenguajes admitidos por la API que viene con el SDK son *C/C++* e *Java*. Los Wrappers en *Python*, *C#* e *Delphi* se pueden proporcionar bajo demanda.

Para preguntas o soporte para otros idiomas, comuníquese con el Soporte Técnico de Pumatronix.

6.1 Limitaciones conocidas

Las limitaciones conocidas de la biblioteca *JidoshaLight* son las siguientes:

1) Cuando la biblioteca se carga dinámicamente (*LoadLibrary* en Windows, *dlopen* en Linux), no se puede descargar (*FreeLibrary* y *dlclose*, respectivamente).
2) En Linux, el proceso donde se ejecuta la biblioteca no se puede bifurcar (*fork*) (copia de proceso).
3) En el caso de uso simultáneo y en el mismo proceso de *JidoshaLight* con la biblioteca *Jidosha Portuario* (container) o *Jidosha Ferrocarril* (rail), JidoshaLight debe cargarse en último lugar. Esta limitación se eliminará en una versión futura de las bibliotecas.

6.2 API JidoshaLight C/C++

La API (Application Programming Interface) nativa de la biblioteca está escrita en lenguaje C, lo que facilita la creación de enlaces para su uso en otros lenguajes. Toda la API de C está disponible a través de un conjunto de *headers* dentro de la carpeta de **include** del SDK.

6.2.1 API JidoshaLight C/C++ (Local)

La API local contiene los tipos básicos, definiciones y funciones para el procesamiento local de imágenes. Desde la versión 2.4.4, su contenido se divide entre los archivos *jidosha_light_api_common.h* y *jidosha_light_api.h*.

Para mantener la compatibilidad con versiones anteriores, el *header* '*jidosha_light_api_common.h*' está incluido en '*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	
Descripción	Define los tipos de placas que debe buscar OCR en la imagen.
Miembros	JIDOSHA_LIGHT_VEHICLE_TYPE_CAR: solo matrículas de auto JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO: solo matrículas de motocicleta JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH: ambos tipos de matrícula

enum JidoshaLightMode	
Descripción	Define las estrategias de procesamiento que puede utilizar el algoritmo de lectura de matrículas. La opción *FAST* usa el menor esfuerzo computacional posible para la lectura mientras que *ULTRA_SLOW* usa el más alto. Cuanto mayor sea el nivel de esfuerzo, mayor será la probabilidad de leer la placa.
Miembros	JIDOSHA_LIGHT_VEHICLE_TYPE_CAR: solo matrículas de auto JIDOSHA_LIGHT_MODE_DISABLE: valor reservado para uso futuro. Actualmente tiene el mismo efecto que la opción ULTRA_SLOW JIDOSHA_LIGHT_MODE_FAST: estrategia de procesamiento más rápida, se recomienda su uso para casos donde el tiempo de procesamiento es crítico JIDOSHA_LIGHT_MODE_NORMAL: estrategia de procesamiento moderado, tiene un tiempo de procesamiento y una tasa de reconocimiento más altos que el método FAST

	JIDOSHA_LIGHT_MODE_SLOW: estrategia de procesamiento lento, tiene un tiempo de procesamiento y una tasa de reconocimiento más altos que el método NORMAL JIDOSHA_LIGHT_MODE_ULTRA_SLOW: estrategia de procesamiento súper lento, tiene el tiempo de procesamiento más largo y la tasa de reconocimiento más alta Se debe utilizar necesariamente uno de los modos anteriores. Además, puede seleccionar opcionalmente la estrategia de ubicación utilizada por el algoritmo de lectura de matrículas. La opción LOCALIZATION_MODE_0 es la predeterminada. Las otras opciones actualmente solo afectan el procesamiento de matrículas de Brasil. JIDOSHA_LIGHT_LOCALIZATION_MODE_0: estrategia de ubicación con mayor tiempo de procesamiento y mayor tasa de reconocimiento JIDOSHA_LIGHT_LOCALIZATION_MODE_1: estrategia de ubicación ligeramente más rápida y tasa de reconocimiento ligeramente inferior JIDOSHA_LIGHT_LOCALIZATION_MODE_2: estrategia de localización rápida, con menor tasa de reconocimiento, aplicable solo a matrículas de automóviles, no de motocicletas

enum JidoshaLightCountryCode	
Descripción	Define el código de país admitido por la biblioteca de reconocimiento en formato ISO 3166-1. El uso de un determinado país está limitado por la licencia.
Miembros	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	
Descripción	Define los tipos de formato RAW compatibles con la biblioteca.
Miembros	JIDOSHA_LIGHT_IMG_FMT_XRGB_8888: formato XRGB de 32 bits, con el byte menos significativo utilizado para el canal azul (*Blue*). Se ignora el byte más significativo. JIDOSHA_LIGHT_IMG_FMT_RGB_888: formato RGB de 24 bits, con el byte menos significativo utilizado para el canal azul (*Blue*) JIDOSHA_LIGHT_IMG_FMT_LUMA: formato de 8 bits que contiene solo el canal de luminancia JIDOSHA_LIGHT_IMG_FMT_YUV420: formato YUV no entrelazado de 8 bits con subsampling 4:2:0

struct JidoshaLightConfig	
Descripción	Define los tipos de formato RAW compatibles con la biblioteca.
Miembros	*int configId:* campo reservado para uso futuro, la biblioteca ignora su valor

	int vehicleType: indica el tipo de matrícula que debe buscar el OCR. Ver enum JidoshaLightVehicleType *int processingMode:* indica la estrategia de procesamiento a utilizar. Ver enum JidoshaLightMode *int timeout:* indica el tiempo máximo en milisegundos para el reconocimiento de matrículas. El valor '0' indica que no hay timeout. Un valor distinto de '0' ayuda a mantener bajo el tiempo de procesamiento promedio: el procesamiento se detiene y la función regresa tan pronto como expira el timeout. El valor debe determinarse en función de la resolución de la imagen y la CPU utilizada *int countryCode:* indica el código del país cuya matrícula se quiere reconocer. Ver enum JidoshaLightCountryCode *float minProbPerChar*: valor de '*0.0f* a '*1.0f* utilizado para definir la probabilidad mínima que debe tener un carácter dado en la placa para ser considerado válido (recomendado: 0.85) *int maxLowProbabilityChars*: número máximo de caracteres con probabilidad menor que '*minProbPerChar*' para que la matrícula reconocida se considere válida - una matrícula reconocida como no válida se devuelve como una string vacía '\0' *char lowProbabilityChar*: carácter que se usará para reemplazar aquellos con probabilidad menor que '*minProbPerChar*' *float avgPlateAngle*: ángulo medio de las placas en las imágenes en relación con el eje horizontal *float avgPlateSlant*: ángulo medio de inclinación de los caracteres de las imágenes en relación con el eje vertical *int maxCharHeight*: altura de carácter máxima aceptable, en píxeles *int minCharHeight*: altura de carácter mínima aceptable, en píxeles *int maxCharWidth*: ancho de carácter máximo aceptable, en píxeles *int minCharWidth*: ancho de carácter mínimo aceptable, en píxeles *int avgCharHeight*: altura promedio de los caracteres, en píxeles (valor predeterminado: 20) *int avgCharWidth*: ancho promedio de caracteres, en píxeles (valor predeterminado: 7) *int xRoi[4]* e *int yRoi[4]*: coordenadas x e y de los cuatro puntos de la región de interés de la imagen (ROI - Region Of Interest) en cualquier orden. Se entiende por región de interés un cuadrilátero dentro de la imagen donde se espera encontrar las matrículas a reconocer. El uso de ROI beneficia el tiempo de procesamiento y la tasa de acierto, ya que excluye regiones viales o lugares sin importancia para el proceso de reconocimiento de matrículas. Establecer todas las coordenadas en cero hará que se ignore el ROI y se procese la imagen completa. Valores mayores a las dimensiones de la imagen o negativos dan como resultado la devolución del código de error JIDOSHA_LIGHT_ERROR_INVALID_ROI. Cambiar estos valores provoca el recálculo de la región ROI, lo que afecta el tiempo de procesamiento de la primera imagen después del cambio.

Atención: Las coordenadas de los puntos ROI tienen su origen (0,0) en la esquina superior izquierda de la imagen y se extienden hasta la esquina inferior derecha (ancho-1, alto-1). Entonces, para una imagen de resolución de 800x600, los valores válidos para los puntos de ROI van desde (0,0) a (799,599). También se debe tener en cuenta que los 4 puntos no pueden ser colineales.

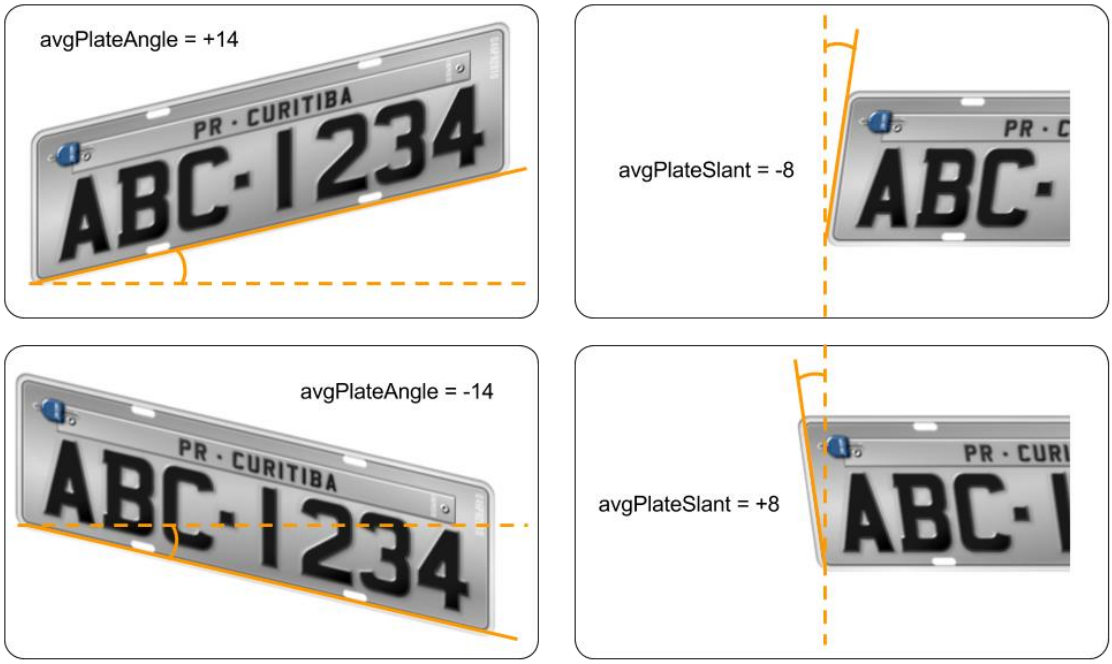


Figura 10 - Cómo calcular los valores avgPlateAngle y avgPlateSlant

struct JidoshaLightRecognitionInfo	
Descripción	El propósito de esta estructura es traer información relacionada con el tiempo de procesamiento de *JidoshaLight*, facilitando el diagnóstico de desempeño. Todos los tiempos se dan en milisegundos.
Miembros	*double totalTime*: tiempo total de procesamiento de la imagen (suma de los otros tiempos). *double localizationTime*: tiempo empleado en localizar la matrícula en la imagen. *double segmentationTime*: tiempo empleado en extraer caracteres de la matrícula. *double classificationTime*: tiempo empleado en el paso de clasificación de caracteres de la placa. *doble loadDecodeTime*: tiempo dedicado a leer y decodificar el archivo de imagen. *int libVersion[3]*: versión de la biblioteca que procesó la imagen. *char libSHA1[41]*: identificador de la biblioteca que procesó la imagen.

struct JidoshaLightRecognition	
Descripción	El propósito de esta estructura es almacenar el resultado del reconocimiento de matrículas, incluyendo: los caracteres de la matrícula, la fiabilidad de cada carácter y las coordenadas de la matrícula en la imagen.
Miembros	*int frameId*: campo reservado para uso futuro, su valor siempre es cero. *char plate[8]*: placa de 7 caracteres terminada en 0, o string vacía si no se encuentra la placa. *float probabilities[7]*: valores de 0,0 a 1,0 que indican la fiabilidad, en forma de probabilidad, de reconocer cada carácter. *int xText* e *int yText*: coordenadas de la esquina superior izquierda de la placa, si se encuentra. *int widthText*: ancho del rectángulo de la placa. *int heightText*: altura del rectángulo de la placa.

	int xChar[7] e *int yChar[7]*: coordenada de la esquina superior izquierda de cada carácter reconocido. *int widthChar[7]*: ancho del rectángulo para cada uno de los caracteres reconocidos. *int heightChar[7]*: altura del rectángulo de cada carácter reconocido. *int textColor*: color del texto de la placa, 0 - oscuro, 1 - claro. *int isMotorcycle*: indica si la matrícula es de moto, 0 - no moto, 1 - moto. *int countryCode*: indica el código de país (en la norma ISO 3166-1) de la matrícula reconocida. Los valores posibles para este campo se definen en la enum JidoshaLightCountryCode. *JidoshaLightRecognitionInfo info*: estructura que contiene información relacionada con el tiempo de procesamiento.

struct JidoshaLightLicenseInfo	
Descripción	Struct utilizada para almacenar información sobre la licencia utilizada por la biblioteca JidoshaLight.
Miembros	*uint64_t serial*: número de serie de la licencia *char customer[64]*: nombre del cliente que compró la licencia *int maxThreads*: número máximo de threads de procesamiento habilitados *int maxConnections*: número máximo de conexiones paralelas habilitadas *int state*: estado de la licencia (ver Códigos de retorno de función) *int ttl*: time-to-live en horas para licencias tipo RTC. Este campo tiene el valor **-1** si la licencia no está caducada

struct JidoshaLightRecognitionList	
Descripción	Tipo opaco utilizado por la biblioteca para devolver una lista de objetos de tipo JidoshaLightRecognition. Las funciones que manipulan la lista insertan nuevos elementos al final de la lista. Para borrar una lista, simplemente destrúyala y cree una nueva. Para evitar pérdidas de memoria, el usuario siempre debe destruir las listas que ya no se utilizan. Este tipo no es thread safe.
Miembros	Ninguno
Métodos relacionados	jidoshaLight_ANPR_createList jidoshaLight_ANPR_duplicateList jidoshaLight_ANPR_destroyList jidoshaLight_ANPR_getListSize jidoshaLight_ANPR_getListElement

struct JidoshaLightImage	
Descripción	Tipo opaco utilizado por la biblioteca para cargar una imagen para ser procesada. Una vez creado, un objeto *JidoshaLightImage* se puede usar para cargar numerosas imágenes, incluso si tienen diferentes formatos. Sin embargo, las llamadas posteriores hacen que se sobrescriba el contenido previamente cargado. El proceso de decode de imágenes se puede posponer al tiempo de procesamiento si el modo *LazyDecode* está habilitado. En esta situación, las funciones de *load* solo almacenan el

	contenido del búfer RAW de la imagen sin realizar ningún procesamiento. Este comportamiento es útil para las aplicaciones cliente-servidor, ya que el costo computacional de decode se delega al servidor. Para evitar pérdidas de memoria, el usuario siempre debe destruir las imágenes que ya no se utilizan. Este tipo no es thread safe.
Miembros	Ninguno
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	
Prototipo de Función	`JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();`
Descripción	Función utilizada para crear una JidoshaLightRecognitionList vacía
Parámetros	Ninguno
Retorno	Un puntero válido al tipo *JidoshaLightRecognitionList* o *NULL* en caso de error.

jidoshaLight_ANPR_duplicateList	
Prototipo de Función	`JidoshaLightRecognitionList* jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);`
Descripción	Función utilizada para duplicar una JidoshaLightRecognitionList
Parámetros	*JidoshaLightRecognitionList* list*: puntero a un objeto *JidoshaLightRecognitionList*
Retorno	Un puntero válido al tipo *JidoshaLightRecognitionList* o *NULL* en caso de error.

jidoshaLight_ANPR_destroyList	
Prototipo de Función	`int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);`
Descripción	Función utilizada para destruir objetos creados por las funciones *jidoshaLight_ANPR_createList* y *jidoshaLight_ANPR_duplicateList*
Parámetros	*JidoshaLightRecognitionList* list*: puntero válido a un objeto *JidoshaLightRecognitionList*
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_getListSize	
Prototipo de Función	`int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);`
Descripción	Función utilizada para leer el número de elementos almacenados dentro de la lista
Parámetros	*JidoshaLightRecognitionList* list*: puntero válido a un objeto *JidoshaLightRecognitionList*
Retorno	Número de elementos presentes en la lista (valor mayor o igual a cero) o -1 en caso de error (**list* no válida).

jidoshaLight_ANPR_getListElement	
Prototipo de Función	`const JidoshaLightRecognition* jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);`
Descripción	Función utilizada para recuperar un puntero al elemento presente en la posición *pos* de la lista. El usuario no puede modificar el contenido del puntero devuelto (const).
Parámetros	*JidoshaLightRecognitionList* list*: puntero válido a un objeto *JidoshaLightRecognitionList* *int pos*: posición del elemento a recuperar en el rango *'[0,ListSize)*
Retorno	Puntero válido a un objeto inmutable de tipo JidoshaLightRecognition o *NULL* en caso de error (**list* no válida o *pos* fuera de rango).

jidoshaLight_ANPR_createImage	
Prototipo de Función	`JidoshaLightImage* jidoshaLight_ANPR_createImage();`
Descripción	Función utilizada para crear una JidoshaLightImage
Parámetros	Ninguno
Retorno	Puntero válido para escribir *JidoshaLightImage* o *NULL* en caso de error.

jidoshaLight_ANPR_duplicateImage	
Prototipo de Función	`JidoshaLightImage* jidoshaLight_ANPR_duplicateImage(JidoshaLightImage* img);`
Descripción	Función utilizada para crear una JidoshaLightImage La imagen duplicada hereda el estado de la imagen original.
Parámetros	*JidoshaLightImage* img*: puntero a un objeto *JidoshaLightImage*
Retorno	Puntero válido para escribir *JidoshaLightImage* o *NULL* en caso de error.

jidoshaLight_ANPR_destroyImage	
Prototipo de Función	`int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);`

Descripción	Función utilizada para destruir objetos creados por las funciones jidoshaLight_ANPR_createImage y jidoshaLight_ANPR_duplicateImage
Parámetros	*JidoshaLightImage* img*: puntero a un objeto *JidoshaLightImage*
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_setImageLazyDecode	
Prototipo de Función	`int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);`
Descripción	Función utilizada para habilitar el modo *LazyDecode* (ver descripción en JidoshaLightImage). El cambio entra en vigencia de inmediato e invalida cualquier imagen cargada previamente.
Parámetros	*JidoshaLightImage* img*: puntero a un objeto *JidoshaLightImage* *int enable*: 0 deshabilitado (default), 1 habilitado
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_loadImageFromFile	

Prototipo de Función	`int jidoshaLight_ANPR_loadImageFromFile (`
	`  JidoshaLightImage* img,`
	`  const char* filename`
	`);`

Descripción	Función utilizada para cargar una *JidoshaLightImage* desde un archivo. Formatos de archivo admitidos: *JPEG, BMP, PNG* y *TIFF*.
Parámetros	*JidoshaLightImage* img*: puntero a un objeto *JidoshaLightImage* *const char* filename*: ruta absoluta al archivo a cargar
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_loadImageFromMemory	

Prototipo de Función	`int jidoshaLight_ANPR_loadImageFromMemory (`
	`  JidoshaLightImage* img,`
	`  const uint8_t* buffer,`
	`  int bufferSize`
	`);`

Descripción	Función utilizada para cargar una *JidoshaLightImage* desde un archivo ya cargado en memoria. Formatos de archivo admitidos: *JPEG, BMP, PNG* y *TIFF*.
Parámetros	*JidoshaLightImage* img*: puntero a un objeto *JidoshaLightImage* *const uint8_t* buffer*: puntero al buffer que contiene el archivo cargado *int bufferSize*: tamaño en bytes del buffer

Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_loadImageFromRawImgFmt	

Prototipo de Función	`int jidoshaLight_ANPR_loadImageFromRawImgFmt (`
	`  JidoshaLightImage* img,`
	`  const uint8_t* buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  JidoshaLightRawImgFmt fmt`
	`);`

Descripción	Función utilizada para cargar una *JidoshaLightImage* desde un buffer que contiene una imagen en formato RAW. Ver formatos admitidos en enum JidoshaLightRawImgFmt
Parámetros	*JidoshaLightImage* img*: puntero válido a un objeto *JidoshaLightImage* *const uint8_t* buffer*: puntero al buffer que contiene la imagen en formato RAW *int width*: ancho en píxeles de la imagen *int height*: altura en píxeles de la imagen *int stride*: tamaño en bytes de una línea de imagen *JidoshaLightRawImgFmt fmt*: formato de imagen
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_fromFile	

Prototipo de Función	`int jidoshaLight_ANPR_fromFile (`
	`  const char* filename,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descripción	Reconoce una matrícula de un archivo de imagen cuya ruta se proporciona en *const char* filename*. Utiliza la configuración definida en *JidoshaLightConfig* config* y devuelve el resultado del reconocimiento en struct *JidoshaLightRecognition* rec*. Si no se puede encontrar una placa en la imagen, el campo *rec->plate* se devuelve vacío. Si se produce un error durante el procesamiento, la struct *JidoshaLightRecognition* rec* se devolverá vacía y la función devolverá un valor distinto de *JIDOSHA_LIGHT_SUCCESS*. Los posibles valores de retorno se definen en la *enum JidoshaLightReturnCode*. Ver formatos admitidos en jidoshaLight_ANPR_loadImageFromFile..
Parámetros	*const char* filename*: ruta al archivo de imagen. *JidoshaLightConfig* config*: puntero a *'struct JidoshaLightConfig'* con la configuración de la biblioteca. Un puntero *'NULL'* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca. *JidoshaLightRecognition* rec*: puntero a la *'struct JidoshaLightRecognition'* donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_fromMemory	

Prototipo de Función	`int jidoshaLight_ANPR_fromMemory (`
	`  const unsigned char* buffer,`
	`  int bufferSize,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descripción	Reconoce una placa de un *buffer* que contiene un archivo de imagen previamente cargado en la memoria. Utiliza la configuración definida en *JidoshaLightConfig* config* y devuelve el resultado del reconocimiento en struct *JidoshaLightRecognition* rec.* Si no se puede encontrar una placa en la imagen, el campo *rec->plate* se devuelve vacío. Si se produce un error durante el procesamiento, la struct *JidoshaLightRecognition* rec* se devolverá vacía y la función devolverá un valor distinto de *JIDOSHA_LIGHT_SUCCESS*. Los posibles valores de retorno se definen en la *enum JidoshaLightReturnCode*. Ver formatos admitidos en jidoshaLight_ANPR_loadImageFromMemory..
Parámetros	*const unsigned char* buffer*: array de bytes que contiene la imagen. *int bufferSize*: tamaño del array de bytes. *JidoshaLightConfig* config*: puntero a *struct JidoshaLightConfig* con la configuración de la biblioteca. Un puntero *NULL* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca. *JidoshaLightRecognition* rec*: puntero a la struct *JidoshaLightRecognition* donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_fromLuma	

Prototipo de Función	`int jidoshaLight_ANPR_fromLuma (`
	`  unsigned char* luma,`
	`  int width,`
	`  int height,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descripción	Reconoce una placa de un búfer que contiene una imagen en formato RAW en escala de grises de 8 bits. Utiliza la configuración definida en *JidoshaLightConfig* config* y devuelve el resultado del reconocimiento en struct *JidoshaLightRecognition* rec*. Si no se puede encontrar una placa en la imagen, el campo *rec->plate* se devuelve vacío. Si se produce un error durante el procesamiento, la struct *JidoshaLightRecognition* rec* se devolverá vacía y la función devolverá un valor distinto de *JIDOSHA_LIGHT_SUCCESS*. Los posibles valores de retorno se definen en la enum *JidoshaLightReturnCode*.
Parámetros	*unsigned char* luma*: array de bytes que contiene la imagen en formato RAW de escala de grises de 8 bits. *int width*: ancho de la imagen *int height*: altura de la imagen. *JidoshaLightConfig* config*: puntero a *struct JidoshaLightConfig* con la configuración de la biblioteca. Un puntero *NULL* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca.

	JidoshaLightRecognition rec*: puntero a la struct *JidoshaLightRecognition* donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_fromRawImgFmt	

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

Descripción	Reconoce una matrícula de un *buffer* que contiene una imagen en uno de los formatos RAW definidos en la *enum JidoshaLightRawImgFmt.* Utiliza la configuración definida en *JidoshaLightConfig* config* y devuelve el resultado del reconocimiento en struct *JidoshaLightRecognition* rec*. Si no se puede encontrar una placa en la imagen, el campo *rec->plate* se devuelve vacío. Si se produce un error durante el procesamiento, la struct *JidoshaLightRecognition* rec* se devolverá vacía y la función devolverá un valor distinto de *JIDOSHA_LIGHT_SUCCESS*. Los posibles valores de retorno se definen en la enum *JidoshaLightReturnCode*. Ver formatos admitidos en jidoshaLight_ANPR_loadImageFromRawImgFmt.
Parámetros	*const unsigned char* buffer*: array de bytes que contiene la imagen en formato RAW. *int width*: ancho de la imagen *int height*: altura de la imagen. *int stride*: tamaño en bytes de una línea de imagen. *JidoshaLightRawImgFmt fmt*: formato de imagen. *JidoshaLightConfig* config*: puntero a *struct JidoshaLightConfig* con la configuración de la biblioteca. Un puntero *NULL* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca. *JidoshaLightRecognition* rec*: puntero a la struct *JidoshaLightRecognition* donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_fromImage	

Prototipo de Función	`int jidoshaLight_ANPR_fromImage(`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descripción	Reconoce una matrícula de una *JidoshaLightImage* previamente cargada. Utiliza la configuración definida en *JidoshaLightConfig* config* y devuelve el resultado del reconocimiento en struct *JidoshaLightRecognition* rec*. Si no se puede encontrar una placa en la imagen, el campo *rec->plate* se devuelve vacío.

	Si se produce un error durante el procesamiento, la struct *JidoshaLightRecognition* rec* se devolverá vacía y la función devolverá un valor distinto de *JIDOSHA_LIGHT_SUCCESS*. Los posibles valores de retorno se definen en la enum *JidoshaLightReturnCode*.
Parámetros	*JidoshaLightImage* img*: puntero a una *JidoshaLightImage* válida *JidoshaLightConfig* config*: puntero a *struct JidoshaLightConfig* con la configuración de la biblioteca. Un puntero *NULL* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca. *JidoshaLightRecognition* rec*: puntero a la struct *JidoshaLightRecognition* donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_ANPR_multi_fromImage	

Prototipo de Función	`int jidoshaLight_ANPR_multi_fromImage (`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config,`
	`  int maxPlates,`
	`  JidoshaLightRecognitionList* list`
	`);`

Descripción	Reconoce varias matrículas de una *JidoshaLightImage* previamente cargada. Utiliza la configuración definida en la *JidoshaLightConfig* config* y agrega reconocimientos de *'maxPlates'* al final de la lista '*JidoshaLightRecognitionList**'. Si el número de placas encontradas es menor que el valor especificado, se agregarán elementos '*JidoshaLightRecognition*' vacíos a la lista hasta que se llene '*maxPlates*'. En caso de error, los elementos *JidoshaLightRecognition* vacíos se agregarán a la lista y la función devolverá un código de retorno que no sea *JIDOSHA_LIGHT_SUCCESS* (ver enum JidoshaLightReturnCode).
Parámetros	*JidoshaLightImage* img*: puntero a una *JidoshaLightImage* válida *JidoshaLightConfig* config*: puntero a *struct JidoshaLightConfig* con la configuración de la biblioteca. Un puntero *NULL* en este parámetro hace que se utilice la configuración predeterminada de la biblioteca. *int maxPlates*: número máximo de placas a reconocer (1 o más) *JidoshaLightRecognitionList* list*: puntero a un objeto JidoshaLightRecognitionList donde se agregarán nuevos *maxPlates* JidoshaLightRecognition.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jidoshaLight_getVersion	
Prototipo de Función	`int jidoshaLight_getVersion(int* major, int* minor, int* release);`
Descripción	Se utiliza para comprobar la versión de la biblioteca, en formato major.minor.release.
Parámetros	*int major, minor, release*: punteros a variables int donde se escribirán los números que componen la versión.
Retorno	Siempre devuelve *JIDOSHA_LIGHT_SUCCESS*.

jidoshaLight_getBuildSHA1	
Prototipo de Función	`const char* jidoshaLight_getBuildSHA1();`
Descripción	Se usa para verificar el SHA1 de la *build* de la biblioteca.
Parámetros	Ninguno
Retorno	Devuelve un puntero al principio de una string terminada en \0 que contiene el valor SHA1 de *build*.

jidoshaLight_getBuildFlags	
Prototipo de Función	`const char* jidoshaLight_getBuildFlags();`
Descripción	Se utiliza para verificar las opciones de *build* de la biblioteca.
Parámetros	Ninguno
Retorno	Devuelve un puntero al principio de una string terminada en \0 que contiene opciones de *build*.

jidoshaLight_isRemoteApi	
Prototipo de Función	`JL_API int jidoshaLight_isRemoteApi();`
Descripción	Comprueba si la biblioteca de la aplicación implementa el procesamiento local o remoto.
Parámetros	Ninguno
Retorno	Devuelve 0 si la API se implementa con acceso local o diferente de este valor si el procesamiento es remoto.

jidoshaLight_getLicenseInfo	
Prototipo de Función	`int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info)`
Descripción	Función utilizada para leer la información de licencia utilizada por la biblioteca JidoshaLight.
Parámetros	*JidoshaLightLicenseInfo* info*: puntero a una struct JidoshaLightLicenseInfo
Retorno	Devuelve JIDOSHA_LIGHT_SUCCESS en caso de éxito.

jidoshaLight_getReturnCodeString	
Prototipo de Función	`const char* jidoshaLight_getReturnCodeString(int rc)`

Descripción	Función utilizada para convertir un código de retorno de biblioteca en una string C.
Parámetros	*int rc*: algún código de retorno definido en la enum JidoshaLightReturnCode y enum JidoshaLightReturnCodeNetwork
Retorno	String que representa el código de error.

6.2.1.5 Códigos de retorno de función

Los códigos de retorno de las funciones están relacionados con el proceso de reconocimiento ('*enum JidoshaLightReturnCode*') o el proceso remoto ('*enum JidoshaLightReturnCodeNetwork*`'). Códigos:

- JIDOSHA_LIGHT_ERROR_FILE_NOT_FOUND: devuelto por las funciones jidoshaLight_ANPR_fromFile e jidoshaLight_ANPR_loadImageFromFile cuando la ruta del archivo especificado no existe.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE: devuelto por las funciones de procesamiento y carga de imágenes. Ocurre cuando la imagen pasada está dañada.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_TYPE: devuelto por las funciones jidoshaLight_ANPR_fromFile, jidoshaLight_ANPR_fromMemory y funciones relacionadas con la carga de JidoshaLightImage. Ocurre cuando se intenta procesar una imagen de un formato no compatible.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_SIZE: devuelto por las funciones jidoshaLight_ANPR_fromFile, jidoshaLight_ANPR_fromMemory y funciones relacionadas con la carga de JidoshaLightImage. Ocurre al intentar procesar una imagen cuyo tamaño supera los límites máximos admitidos por la biblioteca (ARM Zynq: 1280x960px, Outros: 2500x2500px).
- JIDOSHA_LIGHT_ERROR_INVALID_PROPERTY: devuelto por todas las funciones que tienen argumentos. Ocurre cuando el argumento no es válido. En el caso de funciones que reciben punteros, este código se devuelve cuando el argumento es **NULL** (excepto en los casos en que **NULL** es un valor válido para el argumento).
- JIDOSHA_LIGHT_ERROR_COUNTRY_NOT_SUPPORTED: devuelto por funciones **ANPR** cuando el código de país proporcionado en la estructura de configuración no es compatible con la biblioteca.
- JIDOSHA_LIGHT_ERROR_API_CALL_NOT_SUPPORTED: devuelto cuando una función API no está disponible para una plataforma determinada.
- JIDOSHA_LIGHT_ERROR_INVALID_ROI: devuelto cuando se proporciona un ROI no válido. Consulte la descripción de la struct JidoshaLightConfig para obtener más información.
- JIDOSHA_LIGHT_ERROR_INVALID_HANDLE: devuelto cuando el *handle* pasado a la función no se inicializó correctamente.
- JIDOSHA_LIGHT_ERROR_API_CALL_HAS_NO_EFFECT: devuelto cuando una función API no tuvo efecto cuando se ejecutó. Puede ocurrir cuando hay precedencia entre llamadas.
- JIDOSHA_LIGHT_ERROR_LICENSE_INVALID: devuelto por las funciones **ANPR** cuando la *hardkey* no está presente o tiene problemas. Póngase en contacto con Pumatronix Equipamentos Eletrônicos para más información.

- JIDOSHA_LIGHT_ERROR_LICENSE_EXPIRED: devuelto por las funciones **ANPR** cuando una *hardkey* de tipo demo ha caducado. Póngase en contacto con Pumatronix Equipamentos Eletrônicos para más información.
- JIDOSHA_LIGHT_ERROR_LICENSE_MAX_THREADS_EXCEEDED: devuelto por funciones **ANPR** cuando el número máximo de threads concurrentes supera lo permitido por la licencia.
- JIDOSHA_LIGHT_ERROR_LICENSE_UNTRUSTED_RTC: devuelto por las funciones **ANPR** cuando una licencia con fecha de caducidad no tiene disponible una referencia de fecha/hora fiable.
- JIDOSHA_LIGHT_ERROR_OTHER: devuelto cuando ocurre un error inesperado. Póngase en contacto con Pumatronix Equipamentos Eletrônicos para soporte.
- JIDOSHA_LIGHT_ERROR_SERVER_CONNECT_FAILED: devuelto cuando una llamada a la API remota no logra conectarse al servidor.
- JIDOSHA_LIGHT_ERROR_SERVER_DISCONNECTED: devuelto cuando una sesión remota con el servidor se cerró inesperadamente.
- JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_TIMEOUT: devuelto cuando se descartó una solicitud en el servidor debido al timeout.
- JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_FULL: devuelto cuando se descarta una solicitud en el servidor por falta de espacio en la cola.
- JIDOSHA_LIGHT_ERROR_SOCKET_IO_ERROR: devuelto cuando se produjo un error de I/O de red en una sesión remota con el servidor.
- JIDOSHA_LIGHT_ERROR_SOCKET_WRITE_FAILED: devuelto cuando hay un error en el envío de mensajes entre el cliente y el servidor remoto.
- JIDOSHA_LIGHT_ERROR_SOCKET_READ_TIMEOUT: devuelto cuando hay un error en la recepción de mensajes entre el cliente y el servidor remoto.
- JIDOSHA_LIGHT_ERROR_SOCKET_INVALID_RESPONSE: devuelto cuando se recibió un mensaje no válido.
- JIDOSHA_LIGHT_ERROR_HANDLE_QUEUE_FULL: devuelto cuando la cola de solicitudes pendientes ha alcanzado el máximo para un *handle* asíncrono determinado.
- JIDOSHA_LIGHT_ERROR_SERVER_CONN_LIMIT_REACHED: devuelto al intentar conectarse a un servidor con el número máximo de sesiones abiertas.
- JIDOSHA_LIGHT_ERROR_SERVER_VERSION_NOT_SUPPORTED: devuelto cuando la versión del servidor no es compatible con la versión de la biblioteca del cliente.
- JIDOSHA_LIGHT_ERROR_SERVER_NOT_READY: devuelto cuando el servidor se está iniciando, pero aún no está listo para procesar imágenes. El cliente debe esperar e intentar volver a conectarse.

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

La API Remota Síncrona amplía la API local, lo que le permite configurar un servidor remoto para procesar imágenes de forma remota en lugar de localmente. Debe usarse junto con la biblioteca *libjidoshaLightRemote.so*.

Las llamadas *jidoshaLight_ANPR** definidas en la API local siguen siendo válidas, pero el procesamiento se vuelve remoto cuando la aplicación se vincula con *libjidoshaLightRemote.so*.

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

6.2.2.1 Métodos

jidoshaLight_setRemoteSyncServerIp	

Prototipo de Función	`JL_API int jidoshaLight_setRemoteSyncServerIp(`
	`  const char* ip,`
	`  unsigned int port`
	`);`

Descripción	Configura globalmente la dirección IP y el puerto TCP utilizados para conectarse a un servidor de reconocimiento de matrículas. La sesión se establece y se cierra con cada llamada de reconocimiento.
Parámetros	*const char* ip*: string que contiene la dirección IP del servidor. *int port*: entero que contiene el puerto TCP del servidor.
Retorno	Código de retorno JIDOSHA_LIGHT_SUCCESS en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

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

La API Remota Asíncrona amplía la API Local, lo que le permite configurar un servidor remoto que procesará imágenes de forma remota en lugar de procesarlas localmente. Debe usarse junto con la 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*	
Descripción	El propósito de esta estructura es almacenar el objeto cliente de un servidor de reconocimiento de matrículas.
Miembros	Ninguno

typedef void JCallback	
Descripción	El propósito de este tipo es definir el formato de *callback* del usuario para recibir eventos del servidor.
Miembros	*struct JidoshaLightRecognition rec*: struct donde se almacenará el resultado del reconocimiento *int rc*: solicitar código de retorno (ver Códigos de retorno de funciones) *uint8_t* buffer*: puntero a la imagen donde se realizó el reconocimiento (este puntero solo es válido durante la ejecución de *callback*) *unsigned int bufferSize*: tamaño de la imagen *void* arg*: puntero a la estructura opaca proporcionada por el usuario al crear el *handle*

struct JidoshaLightClientConfig	
Descripción	El propósito de esta estructura es definir los parámetros de conexión entre el cliente y el servidor.
Miembros	*int queueSize*: tamaño máximo de solicitudes pendientes para este *handle*. *const char* ip*: string que contiene la dirección IP del servidor. *int port*: entero que contiene el puerto TCP del servidor. *JCallback callback*: función designada para procesar los resultados generados por el servidor. *void* arg*: puntero opaco a la estructura de datos del usuario utilizada para manejar eventos del servidor. Este puntero se pasa como un parámetro de *callback* del usuario.

struct JidoshaLightServerInfo	
Descripción	Struct utilizada para almacenar información de licencia y versión de un servidor JidoshaLight.
Miembros	*JidoshaLightLicenseInfo license*: *estructura que contiene información de la licencia del servidor* - ver struct JidoshaLightLicenseInfo

	int major: valor principal de la versión de la biblioteca utilizada por el servidor *int minor*: valor menor de la versión de la biblioteca utilizada por el servidor *int release*: valor de release de la versión de la biblioteca utilizada por el servidor

6.2.3.2 Métodos

jl_async_create_handle	

Prototipo de Función	`JidoshaLightHandle* jl_async_create_handle(`
	`  JidoshaLightClientConfig* config`
	`);`

Descripción	Crea el *handle* de un cliente asíncrono para conectarse a un servidor de reconocimiento de matrículas.
Parámetros	*JidoshaLightClientConfig* config*: configuración para este *handle*.
Retorno	Devuelve un puntero al *handle* de tipo *JidoshaLightHandle* o *NULL* en caso de error.

jl_async_destroy_handle	

Prototipo de Función	`int jl_async_destroy_handle(`
	`  JidoshaLightHandle* handle`
	`);`

Descripción	Desasigna el *handle* de un cliente asíncrono, cerrando la conexión con el servidor de reconocimiento de matrículas.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jl_async_connect	

Prototipo de Función	`int jl_async_connect(`
	`  JidoshaLightHandle* handle`
	`);`

Descripción	Establece una sesión con el servidor de reconocimiento de matrículas para un *handle* determinado. Esta función se bloquea hasta que se establece la conexión o se agota el *timeout*.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jl_async_connect_info	

Prototipo de Función	`int jl_async_connect_info(`
	`  JidoshaLightHandle* handle,`
	`  JidoshaLightServerInfo* info`
	`);`

Descripción	Tiene la misma funcionalidad que la función jl_async_connect pero toma un parámetro adicional para recibir información sobre la licencia y la versión del servidor.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle. *JidoshaLightServerInfo* info*: Puntero a una struct JidoshaLightServerInfo
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

jl_async_get_localqueue_size	

Prototipo de Función	`int jl_async_get_localqueue_size(`
	`  JidoshaLightHandle* handle`
	`);`

Descripción	Devuelve el tamaño de la cola de solicitudes pendientes en el lado del cliente para un *handle* determinado.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle
Retorno	Devuelve el número de solicitudes pendientes en la cola local (cliente).

jl_async_ANPR_fromFile	

Prototipo de Función	`int jl_async_ANPR_fromFile(`
	`  JidoshaLightHandle* handle,`
	`  const char* filename,`
	`  JidoshaLightConfig* config`
	`);`

Descripción	Ver descripción del método jidoshaLight_ANPR_fromFile
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *const char* filename*: Ver descripción del método jidoshaLight_ANPR_fromFile *JidoshaLightConfig* config*: Ver descripción del método jidoshaLight_ANPR_fromFile
Retorno	Ver descripción del método jidoshaLight_ANPR_fromFile

jl_async_ANPR_fromMemory	

Prototipo de Función	`int jl_async_ANPR_fromMemory(`
	`  JidoshaLightHandle* handle,`
	`  const unsigned char* buffer,`
	`  unsigned int bufferSize,`
	`  JidoshaLightConfig* config`
	`);`

Descripción	Ver descripción del método jidoshaLight_ANPR_fromMemory
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *const unsigned char* buffer*: Ver la descripción del método jidoshaLight_ANPR_fromMemory *unsigned int bufferSize*: Ver la descripción del método jidoshaLight_ANPR_fromMemory *JidoshaLightConfig* config*: Ver la descripción del método jidoshaLight_ANPR_fromMemory
Retorno	Ver descripción del método jidoshaLight_ANPR_fromMemory

jl_async_ANPR_fromLuma	

Prototipo de Función	`int jl_async_ANPR_fromLuma(`
	`  JidoshaLightHandle* handle,`
	`  unsigned char* luma,`
	`  int width,`
	`  int height,`
	`  JidoshaLightConfig* config`
	`);`

Descripción	Ver descripción del método jidoshaLight_ANPR_fromLuma
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *unsigned char* luma*: Ver la descripción del método jidoshaLight_ANPR_fromLuma *int width*: Ver la descripción del método jidoshaLight_ANPR_fromLuma *int height*: Ver la descripción del método jidoshaLight_ANPR_fromLuma *JidoshaLightConfig* config*: Ver la descripción del método jidoshaLight_ANPR_fromLuma
Retorno	Ver la descripción del jidoshaLight_ANPR_fromLuma

jl_async_ANPR_fromRawImgFmt	

Prototipo de Función	`int jl_async_ANPR_fromRawImgFmt (`
	`  JidoshaLightHandle* handle,`
	`  const unsigned char* buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  JidoshaLightRawImgFmt fmt,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descripción	Ver la descripción del jidoshaLight_ANPR_fromRawImgFmt
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *const unsigned char* buffer*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *int width*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *int height*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *int stride*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightRawImgFmt fmt*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightConfig* config*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightRecognition* rec*: Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt
Retorno	Ver la descripción del método jidoshaLight_ANPR_fromRawImgFmt

jl_async_ANPR_fromImage	
Prototipo de Función	`int jl_async_ANPR_fromImage (` `  JidoshaLightHandle* handle,`

	`JidoshaLightImage* img,`
	`JidoshaLightConfig* config`
	`);`

Descripción	Ver la descripción del método jidoshaLight_ANPR_fromImage.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *JidoshaLightImage* img*: Ver la descripción del método jidoshaLight_ANPR_fromImage *JidoshaLightConfig* config*: Ver la descripción del método jidoshaLight_ANPR_fromImage
Retorno	Ver la descripción del método jidoshaLight_ANPR_fromImage.

jl_async_ANPR_multi_fromImage	

Prototipo de Función	`int jl_async_ANPR_multi_fromImage (`
	`JidoshaLightHandle* handle,`
	`JidoshaLightImage* img,`
	`JidoshaLightConfig* config,`
	`int maxPlates`
	`);`

Descripción	Reconoce varias matrículas de una JidoshaLightImage previamente cargada, lo que genera múltiples llamadas a la función de *callback*. Un conjunto de reconocimientos pertenecientes a la misma imagen puede ser identificado por el campo *int frameId* de la struct JidoshaLightRecognition Ver la descripción del método jidoshaLight_ANPR_multi_fromImage para obtener más detalles.
Parámetros	*JidoshaLightHandle* handle*: Puntero al *handle* creado por jl_async_create_handle *JidoshaLightImage* img*: Ver la descripción del método jidoshaLight_ANPR_multi_fromImage. *JidoshaLightConfig* config*: Ver la descripción del método jidoshaLight_ANPR_multi_fromImage. *int maxPlates*: Ver la descripción del método jidoshaLight_ANPR_multi_fromImage
Retorno	Ver la descripción del método jidoshaLight_ANPR_multi_fromImage.

6.2.4 API JidoshaLight C/C++ (Servidor)

La API Servidor amplía la API Local, lo que le permite crear y configurar un servidor de lectura de matrículas para usar con las APIs remotas. Debe usarse junto con la 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	
Descripción	El propósito de esta estructura es almacenar el objeto del servidor de reconocimiento de matrículas.
Miembros	Ninguno

struct JidoshaLightServerConfig	
Descripción	El propósito de esta estructura es configurar el comportamiento de la biblioteca cuando actúa como servidor de reconocimiento de matrículas.
Miembros	*int port*: número de puerto TCP utilizado para el intercambio de mensajes. *int conns*: número de conexiones simultáneas de clientes aceptadas por el servidor. *int threads*: número de threads de procesamiento paralelos iniciados por el servidor. *int threadQueueSize*: tamaño máximo de la cola de solicitudes para cada thread de procesamiento. *int queueTimeout*: tiempo máximo de espera de una solicitud en la cola de procesamiento en milisegundos (ms). El valor *0* indica que no hay timeout.

6.2.4.2 Métodos

jidoshaLightServer_create	

Prototipo de Función	`JidoshaLightServer* jidoshaLightServer_create(`
	`  JidoshaLightServerConfig* serverConfig`
	`);`

Descripción	Crea una instancia de servidor de reconocimiento de matrículas. Utiliza la struct de configuración señalada por *JidoshaLightServerConfig* serverConfig* y devuelve el puntero a un *handler* de tipo *JidoshaLightServer*.
Parámetros	*serverConfig*: puntero para estructurar la *struct JidoshaLightServerConfig* con la configuración del servidor
Retorno	Devuelve un puntero al *handle* de tipo *JidoshaLightServer* o *NULL* en caso de error.

jidoshaLightServer_destroy	

Prototipo de Función	`int jidoshaLightServer_destroy(`
	`  JidoshaLightServer* handler`
	`);`

Descripción	Desasigna la instancia del servidor identificada por su *handler*.
Parámetros	*handler*: Puntero a la instancia del servidor.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* en caso de éxito, otro código de lo contrario (ver Códigos de retorno de función)

6.3 API JidoshaLight Java

La API de Java de la biblioteca JidoshaLight tiene variaciones entre las versiones de Linux y Android™ del SDK.

La versión de Linux es un *wrapper* simple sobre la API C, mientras que la versión de Android™ tiene funciones de procesamiento especializadas que se adaptan mejor a este entorno de desarrollo. Los métodos específicos para una u otra plataforma se especifican en la descripción del 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	
Descripción	Tiene la misma funcionalidad que la struct JidoshaLightImage de la API C.

Métodos públicos	`public JidoshaLightImage();`
	Construye un nuevo objeto de tipo `*JidoshaLightImage*`. Si falla la asignación del *handle* nativo, genera una `*RuntimeException*`.
	Para evitar pérdidas de memoria, cada objeto `*JidoshaLightImage*` creado debe ser destruido explícitamente por el usuario usando la función `*destroy()*`.
	
	`public JidoshaLightImage duplicate();`
	Duplica un objeto de tipo `*JidoshaLightImage*` previamente creado y cargado en la memoria. El nuevo objeto debe ser destruido por el usuario mediante la función `*destroy()*`.
	
	`public int destroy();`
	Libera la memoria asignada por el objeto.
	
	`public int setLazyDecode(boolean enable);`
	Ver struct JidoshaLightImage de la API C.
	
	`public int loadFromFile(String filename);`
	Ver struct JidoshaLightImage de la API C.
	
	`public int loadFromMemory(byte[] buffer);`
	Ver struct JidoshaLightImage de la API C.

	`public int loadFromRawImgFmt(byte[] buffer, int width, int height, int stride, int fmt);` Ver struct JidoshaLightImage de la API C.

class JidoshaLight.Config	
Descripción	Tiene las mismas funcionalidades que la struct JidoshaLightConfig de la API C.
Miembros	*int vehicleType*: indica el tipo de matrícula que debe buscar el OCR. Los valores posibles para este campo son: • 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 la estrategia de procesamiento adoptada por el algoritmo de reconocimiento. Los valores posibles para este campo son: • 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 de los cuatro puntos de la región de interés de la imagen (ROI - Region Of Interest). Ver la API C para obtener más información.

class JidoshaLight.Recognition	
Descripción	Concatena la funcionalidad de los tipos struct JidoshaLightRecognition y struct JidoshaLightRecognitionInfo de la API C.
Miembros	*String plate*: string que contiene los caracteres de la placa reconocida o vacía si no se encontró la placa. *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	
Descripción	Tipo utilizado por la función getLicenseInfo para devolver información sobre la licencia de la biblioteca.
Miembros	*String serial*: serial number de la licencia en decimal *String customer*: nombre del cliente que compró la licencia *int maxThreads*: número máximo de threads de procesamiento habilitados *int maxConnections*: número máximo de conexiones paralelas habilitadas *int state*: estado de la licencia (ver Códigos de retorno de función) *int ttl*: time-to-live en horas para licencias tipo RTC. Este campo tiene el valor **-1** si la licencia no está caducada

class JidoshaLight.Version	
Descripción	Tipo utilizado por la función *getVersion* para devolver la versión de la biblioteca.
Miembros	*int major*: valor principal de la versión. *int minor*: valor de versión menor. *int release*: valor de release de la versión.

6.3.1.2 Métodos

JidoshaLight.ANPR_fromImage [LINUX e ANDROID]	

Prototipo de Función	`public static native int ANPR_fromImage(`
	`  JidoshaLightImage img,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_ANPR_fromImage de la API C.
Parámetros	*img*: objeto de tipo JidoshaLightImage que contiene la imagen a reconocer. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Pasar '*null*' en este parámetro implica usar la configuración predeterminada de la biblioteca.

	rec: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función).

JidoshaLight.ANPR_multi_fromImage [LINUX e ANDROID]	

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

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_ANPR_multi_fromImage de la API C.
Parámetros	*img*: objeto de tipo JidoshaLightImage que contiene la imagen a reconocer. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Pasar '*null*' en este parámetro implica usar la configuración predeterminada de la biblioteca. *maxPlates*: número máximo de placas a reconocer en la imagen (1 a 8) *recList*: lista de objetos de tipo JidoshaLight.Recognition donde se almacenarán los resultados de la lectura. Tiene un tamaño igual a *maxPlates* si la función regresa con éxito.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función).

JidoshaLight.ANPR_fromFile [LINUX]	

Prototipo de Función	`public static native int ANPR_fromFile(`
	`  String filename,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_ANPR_fromFile de la API C.
Parámetros	*filename*: string que contiene la ruta al archivo de imagen que se va a reconocer. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Pasar *'null'* en este parámetro implica usar la configuración predeterminada de la biblioteca. *rec*: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función).

JidoshaLight.ANPR_fromMemory [LINUX]	

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

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_ANPR_fromMemory de la API C.
Parámetros	*buffer:* array de bytes que contiene la imagen. *bufferSize*: tamaño del array de bytes. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto '*null* en este parámetro implica el uso de la configuración predeterminada de la biblioteca. *rec*: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función).

JidoshaLight.ANPR_fromLuma [LINUX]	

Prototipo de Función	`public static native int ANPR_fromLuma(`
	`  byte[] luma,`
	`  int width,`
	`  int height,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_ANPR_fromLuma de la API C.
Parámetros	*luma*: array de bytes que contiene la imagen en formato RAW en escala de grises de 8 bits. *width*: ancho de la imagen. *height*: altura de la imagen. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto '*null* en este parámetro implica el uso de la configuración predeterminada de la biblioteca. *rec*: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función).

JidoshaLight.ANPR_fromBitmap [ANDROID]	

Prototipo de Función	`public static native int ANPR_fromBitmap (`
	`  Bitmap bitmap,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descripción	Reconoce una placa de matrícula del objeto '*bitmap*' usando la configuración presente en '*config*'. El resultado del reconocimiento se devuelve en '*rec*'. Si se produce un error en el proceso de reconocimiento, el objeto '*rec*' contendrá una string vacía como placa y la función devolverá un valor distinto de '*JidoshaLight.SUCCESS*'. Los posibles valores de retorno se definen en la clase '*JidoshaLight*' y contienen el prefijo '*ERROR_*'.
Parámetros	*bitmap*: objeto de tipo '*android.graphics.Bitmap*' que contiene la imagen a reconocer en formato '*ARGB8888*'. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto '*null* en este parámetro implica el uso de la configuración predeterminada de la biblioteca.

	rec: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función)

JidoshaLight.ANPR_fromUri [ANDROID]	

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

Descripción	Reconoce una matrícula a partir del *'Uri'* de un archivo de imagen. Internamente, este método llama a la función *ANPR_fromBitmap*. El resultado del reconocimiento se devuelve en '*rec*'. Si se produce un error en el proceso de reconocimiento, el objeto '*rec*' contendrá una string vacía como placa y la función devolverá un valor distinto de '*JidoshaLight.SUCCESS*'. Los posibles valores de retorno se definen en la clase '*JidoshaLight*' y contienen el prefijo '*ERROR_*'.
Parámetros	*context*: objeto de tipo '*android.content.Context*' que contiene el contexto de la Activity. uri: objeto de tipo '*android.net.Uri*' que contiene el '*url*' de la imagen a reconocer. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto '*null*' en este parámetro implica el uso de la configuración predeterminada de la biblioteca. *rec*: objeto de tipo JidoshaLight.Recognition donde se almacenará el resultado de la lectura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función)

JidoshaLight.getAndroidFingerprint [ANDROID]	
Prototipo de Función	`static native String getAndroidFingerprint(Activity androidActivity);`
Descripción	Devuelve el identificador único generado al instalar la biblioteca.
Parámetros	*androidActivity*: objeto de tipo '*android.app.Activity*' que contiene la referencia a la Activity principal de la aplicación.
Retorno	String que contiene el identificador único de la instalación necesario para generar el archivo de licencia. Esta string no debe cambiarse.

JidoshaLight.getLicenseFromServer [ANDROID]	
Prototipo de Función	`static native int getLicenseFromServer(Activity activity, String savePath, String user, String key);`
Descripción	Solicita un archivo de licencia del servidor de licencias de Pumatronix.
Parámetros	*activity*: objeto de tipo '*android.app.Activity*' que contiene la referencia a la Activity principal de la aplicación. *savePath*: ruta donde se debe guardar el archivo de licencia recibido en caso de éxito. *user*: usuario que se utilizará en la solicitud o '*null*' en caso contrario.

	key: clave que se utilizará en la solicitud o '*null* en caso contrario.
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 Consulte la muestra de Android para obtener más información.

JidoshaLight.setLicenseFromData [ANDROID]	
Prototipo de Función	`static native int setLicenseFromData(Activity androidActivity, byte[] data, int dataSize);`
Descripción	Este método se utiliza para configurar el archivo de licencia de la biblioteca a partir del contenido del *buffer* `*byte[] data*` .
Parámetros	*androidActivity*: objeto de tipo 'android.app.Activity' que contiene la referencia a la Activity principal de la aplicación. *data*: buffer con el contenido del archivo de licencia. *dataSize*: tamaño del archivo de licencia - `*data.len*`
Retorno	Código de retorno *JidoshaLight.SUCCESS* en caso de éxito, otro código en caso contrario (ver Códigos de retorno de función)

JidoshaLight.getVersion	
Prototipo de Función	`public static native int getVersion(Version version);`
Descripción	Devuelve la versión de la biblioteca en formato major.minor.release.
Parámetros	*version*: objeto de tipo JidoshaLight.Version con número de versión
Retorno	Siempre devuelve *JidoshaLight.SUCCESS*.

JidoshaLight.getBuildSHA1	
Prototipo de Función	`String getBuildSHA1();`
Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildSHA1 de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene el valor SHA1 de *build*.

JidoshaLight.getBuildFlags	
Prototipo de Función	`String getBuildFlags();`
Descripción	Tiene el mismo comportamiento que la función JidoshaLight_getBuildFlags de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene los *flags* de *build* de la biblioteca.

JidoshaLight.getLicenseInfo	
Prototipo de Función	`public static native int getLicenseInfo(LicenseInfo info);`
Descripción	Función utilizada para leer la información de licencia utilizada por la biblioteca JidoshaLight.
Parámetros	*info*: objeto de tipo JidoshaLight.LicenseInfo
Retorno	Devuelve *JIDOSHA_LIGHT_SUCCESS* en caso de éxito.

6.3.1.3 Códigos de retorno de función

Los códigos devueltos por las funciones de la biblioteca JidoshaLight se definen como atributos '*public static final int*' dentro de la clase *JidoshaLight*. Los códigos devueltos en las versiones de Linux y ANDROID del SDK son:

- JidoshaLight.ERROR_FILE_NOT_FOUND: devuelto por las funciones '*ANPR_fromFile*' y '*ANPR_fromUri*' cuando la ruta del archivo especificado no existe.
- JidoshaLight.ERROR_INVALID_IMAGE: devuelto por las funciones '*ANPR*'. Ocurre cuando la imagen pasada está dañada.
- JidoshaLight.ERROR_INVALID_IMAGE_TYPE: devuelto por las funciones '*ANPR*'. Ocurre cuando se intenta procesar una imagen de un formato no compatible. La versión de Android de la API no devuelve este código de error.
- JidoshaLight.ERROR_INVALID_PROPERTY: devuelto por todas las funciones que tienen argumentos. Ocurre cuando el argumento no es válido.
- JidoshaLight.ERROR_COUNTRY_NOT_SUPPORTED: devuelto por las funciones '*ANPR*' cuando la biblioteca no admite el código de país proporcionado en la estructura de configuración.
- JidoshaLight.ERROR_API_CALL_NOT_SUPPORTED: devuelto cuando una función API no está disponible para una plataforma determinada.
- JidoshaLight.ERROR_INVALID_ROI: devuelto cuando se proporciona un ROI no válido. Consulte la descripción de *'struct JidoshaLightConfig'* para obtener más información.
- JidoshaLight.ERROR_INVALID_*HANDLE*: devuelto cuando el *handle* pasado a la función no se inicializó correctamente.
- JidoshaLight.ERROR_API_CALL_HAS_NO_EFFECT: devuelto cuando una función API no tuvo efecto cuando se ejecutó. Puede ocurrir cuando hay precedencia entre llamadas.

- JidoshaLight.ERROR_LICENSE_INVALID: devuelto por las funciones '*ANPR*' cuando la licencia suministrada no es válida (para licencias tipo *hardkey*, significa que no está conectado o tiene problemas). Póngase en contacto con Pumatronix Equipamentos Eletrônicos para más información.
- JidoshaLight.ERROR_LICENSE_EXPIRED: devuelto por las funciones '*ANPR*' cuando el período de uso de la licencia ha expirado. Este tipo de error solo ocurre con licencias de tipo demo. Póngase en contacto con Pumatronix Equipamentos Eletrônicos para más información.
- JidoshaLight.ERROR_LICENSE_MAX_THREADS_EXCEEDED: devuelto por las funciones '*ANPR*' cuando el número máximo de threads simultáneos supera lo permitido por la licencia.
- JidoshaLight.ERROR_LICENSE_UNTRUSTED_RTC: devuelto por las funciones '*ANPR*' cuando una licencia con fecha de uso limitada no tiene disponible una referencia de fecha/hora fiable.
- JidoshaLight.ERROR_OTHER: devuelto cuando ocurre un error inesperado. Póngase en contacto con Pumatronix Equipamentos Eletrônicos para soporte.

6.3.2 API JidoshaLight Java (Remota Asíncrona)

Nota: Todas las funciones de la API remota asíncrona están disponibles para Android y 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	
Descripción	El propósito de esta estructura es almacenar el objeto cliente de un servidor de reconocimiento de matrículas.
Miembros	*queueSize*: tamaño máximo de solicitudes pendientes para el *handle*. *ip*: string que contiene la dirección IP del servidor. *port*: entero que contiene el puerto TCP del servidor.

interface JidoshaLightRemote.CallBacks	
Descripción	Interfaz que define el formato de callback para recibir eventos del servidor.
Miembros	*on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer)* • *rec*: objeto que contiene el resultado del reconocimiento • *code*: código de retorno de solicitud • *buffer*: buffer con la imagen utilizada en el reconocimiento

class JidoshaLightRemote.ServerInfo	
Descripción	Struct utilizada para almacenar información de licencia y versión de un servidor JidoshaLight.

Miembros	*JidoshaLight.LicenseInfo license*: información sobre la licencia utilizada por el servidor *JidoshaLight.Version version*: versión de la biblioteca del servidor

6.3.2.2 Métodos

JidoshaLightRemote.create_*handle*	
Prototipo de Función	`long create_handle (Config config, CallBacks callbacks);`
Descripción	Crea el *handle* de un cliente asíncrono para conectarse a un servidor de reconocimiento de matrículas. Se debe realizar una llamada a *destroy_handle* para liberar los recursos asignados.
Parámetros	Un objeto '*JidoshaLightRemote.Config*' que contiene los parámetros de configuración del servidor y un objeto que implementa la interfaz '*JidoshaLightRemote.CallBacks*'.
Retorno	En caso de éxito, devuelve la dirección de memoria del *handle* creado. De lo contrario, devuelve '*0*'.

JidoshaLightRemote.destroy_*handle*	
Prototipo de Función	`int destroy_handle (long handle);`
Descripción	Libera los recursos asignados al *handle*. Para evitar que un *handle* ya liberado se reutilice incorrectamente, se recomienda que después de llamar a esta función, se asigne *'0'* al valor del *handle*, es decir `mHandle = 0;`.
Parámetros	Un *'long'* que contiene la dirección de memoria de un *handle* '*JidoshaLightRemote*' válido.
Retorno	*JidoshaLight.SUCCESS* en caso de éxito.

JidoshaLightRemote.connect	
Prototipo de Función	`int connect (long handle);`
Descripción	Establece una sesión con el servidor de reconocimiento de matrículas para un *handle* determinado.
Parámetros	*long handle*: long que contiene la dirección de memoria de un *handle* JidoshaLightRemote válido.
Retorno	*JidoshaLight.SUCCESS* en caso de éxito; de lo contrario, consulte los códigos de errors.

JidoshaLightRemote.connect_info	
Prototipo de Función	`int connect_info(long handle, ServerInfo info);`
Descripción	Tiene la misma funcionalidad que la función de connect pero toma un parámetro adicional para recibir información sobre la licencia y la versión del servidor.
Parámetros	*long handle*: long que contiene la dirección de memoria de un *handle* JidoshaLightRemote válido. *ServerInfo* info*: Objeto de tipo JidoshaLightRemote.ServerInfo

Retorno	*JidoshaLight.SUCCESS* en caso de éxito; de lo contrario, consulte los códigos de err.

JidoshaLightRemote.get_localqueue_size	
Prototipo de Función	`int get_localqueue_size (long handle);`
Descripción	Devuelve el tamaño de la cola de solicitudes pendientes en el lado del cliente para un *handle* determinado.
Parámetros	Un `*long*` que contiene la dirección de memoria de un *handle* JidoshaLightRemote válido.
Retorno	Devuelve el número de solicitudes pendientes en la cola local (cliente).

JidoshaLightRemote.ANPR_fromMemory	

Prototipo de Función	`int ANPR_fromMemory (`
	`  long handle,`
	`  byte[] buffer,`
	`  JidoshaLight.Config config`
	`);`

Descripción	Versión remota de la llamada `*JidoshaLight.ANPR_fromMemory*`.
Parámetros	*handle*: long que contiene la dirección de memoria de un *handle* JidoshaLightRemote válido. *buffer*: array de bytes que contiene la imagen a reconocer en formato JPEG, PNG o BMP. *config*: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto *'null'* en este parámetro implica el uso de la configuración predeterminada de la biblioteca.
Retorno	*JidoshaLight.SUCCESS* en caso de éxito; de lo contrario, consulte los códigos de errors.

JidoshaLightRemote.ANPR_fromRawImgFmt	

Prototipo de Función	`int ANPR_fromRawImgFmt (`
	`  long handle,`
	`  byte[] buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  int fmt,`
	`  JidoshaLight.Config config`
	`  );`

Descripción	Envía una solicitud de reconocimiento de matrículas a partir de una imagen en formato RAW.
Parámetros	*handle*: long que contiene la dirección de memoria de un *handle* JidoshaLightRemote válido. *buffer*: array de bytes que contiene la imagen que se va a reconocer en uno de los formatos RAW admitidos (consulte las definiciones en la clase *'JidoshaLight'*). *width*: ancho de la imagen. *height*: altura de la imagen. *stride*: número de bytes por la línea de la imagen *fmt*: formato de imagen (ver definiciones en la clase *'JidoshaLight'*).

	config: objeto de tipo JidoshaLight.Config que contiene la configuración de la biblioteca. Un objeto *'null'* en este parámetro implica el uso de la configuración predeterminada de la biblioteca.
Retorno	*JidoshaLight.SUCCESS* en caso de éxito; de lo contrario, consulte los códigos de errors.

JidoshaLightRemote.getVersion	
Prototipo de Función	`public static native int getVersion(Version version);`
Descripción	Devuelve la versión de la biblioteca en formato major.minor.release.
Parámetros	*version*: objeto de tipo `*Version*` con número de versión
Retorno	Siempre devuelve `*JidoshaLight.SUCCESS*`.

JidoshaLightRemote.getBuildSHA1	
Prototipo de Función	`String getBuildSHA1();`
Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildSHA1 de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene el valor SHA1 de *build*.

JidoshaLightRemote.getBuildFlags	
Prototipo de Función	`String getBuildFlags();`
Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildFlags de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene los *flags* de *build* de la biblioteca.

JidoshaLightRemote.getBuildFlags	
Prototipo de Función	`String getBuildFlags();`
Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildFlags de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene los *flags* de *build* de la biblioteca.

6.3.3 API JidoshaLight Java (Servidor)

Nota: Todas las funciones de la API del servidor están disponibles para Android y 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	
Descripción	Estructura de configuración de un servidor lector de matrículas.
Miembros	*port*: puerto de conexión del servidor. *conns*: número máximo de conexiones simultáneas que el servidor puede aceptar (valor máximo limitado por la licencia) *threads*: número máximo de threads de procesamiento que el servidor puede usar (valor máximo limitado por la licencia). Los threads se comparten entre las conexiones. *threadQueueSize*: tamaño máximo de cola de procesamiento para cada thread. *queueTimeout*: tiempo máximo que una solicitud puede esperar en la cola de procesamiento.

6.3.3.2 Métodos

JidoshaLightServer.create_*handle*	
Prototipo de Función	`long create_handle (Config config);`
Descripción	Cree un *handle* para el servidor de reconocimiento de matrículas e inicialícelo. Se debe realizar una llamada a '*destroy_handle*' para liberar los recursos asignados.
Parámetros	Un objeto `*JidoshaLightServer.Config*` que contiene los parámetros de configuración del servidor.
Retorno	En caso de éxito, devuelve la dirección de memoria del *handle* creado. De lo contrario, devuelve '*0*'.

JidoshaLightServer.destroy_*handle*	
Prototipo de Función	`void destroy_handle (long handle);`
Descripción	Libera los recursos asignados al *handle* y detiene el servidor. Para evitar que un *handle* ya liberado se reutilice incorrectamente, se recomienda que después de llamar a esta función, se asigne *'0'* al valor del *handle*, es decir `mHandle = 0;`.
Parámetros	Un `*long*` que contiene la dirección de memoria de un *handle* JidoshaLightServer válido.
Retorno	`*0*` si no se puede crear el *handle*. De lo contrario, devuelve un valor distinto de cero.

JidoshaLightServer.getVersion	
Prototipo de Función	`public static native int getVersion(Version version);`
Descripción	Devuelve la versión de la biblioteca en formato major.minor.release.
Parámetros	*version*: objeto de tipo `*Version*` con número de versión
Retorno	Siempre devuelve `*JidoshaLight.SUCCESS*`.

JidoshaLightServer.getBuildSHA1	
Prototipo de Función	`String getBuildSHA1();`
Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildSHA1 de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene el valor SHA1 de *build*.

JidoshaLightServer.getBuildFlags	
Prototipo de Función	`String getBuildFlags();`

Descripción	Tiene el mismo comportamiento que la función jidoshaLight_getBuildFlags de la API C.
Parámetros	Ninguno
Retorno	Devuelve una String que contiene los *flags* de *build* de la biblioteca.

6.3.4 API JidoshaLight Java (IO/Mjpeg)

Esta API proporciona un receptor de video en formato MJPEG (Motion JPEG). Este formato de vídeo es muy utilizado por las cámaras IP.

Nota: Todas las funciones de la API IO/Mjpeg están disponibles para Android y 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	
Descripción	Estructura de configuración para una transmisión Mjpeg.
Miembros	*url*: String que contiene la URL de la transmisión Mjpeg en formato `http://<IP>[:PORT]/[PATH]`. *timeout*: intervalo máximo entre fotogramas en milisegundos. Los retrasos superiores al *'timeout'* se consideran como pérdida de conexión. (Valores recomendados: 1000 a 5000). *bufferSize*: número máximo de fotogramas que se pueden poner en cola. Este parámetro debe ser mayor que **0** y preferiblemente igual a **1**. Se deben considerar valores mayores que **1** para los casos en los que llamar a la callback '*frame_cb*' puede llevar más tiempo que la velocidad de fotogramas de la transmisión.

interface Mjpeg.Callbacks	
Descripción	Interfaz que define las callbacks generadas por el flujo Mjpeg. **Nota 1: la ejecución de la callback no puede tardar mucho (ver parámetro '*bufferSize*' `).** **Nota 2: bajo ninguna circunstancia puedes llamar a '*destroy_handle(long handle)*' dentro de una callback.**
Miembros	*void frame_cb(byte[] frame)*: callback llamada cada vez que hay un nuevo frame disponible. El frame viene en formato JPEG. *void error_cb(int code)*: callback llamada cada vez que se produce un error en la transmisión Mjpeg (consulte la definición de error a continuación). Siempre que haya un error de desconexión, el flujo intentará restablecer la conectividad automáticamente. Para interrumpir el proceso, el usuario solo tiene que destruir el *handle*.

6.3.4.2 Métodos

Mjpeg.create_*handle*	

Prototipo de Función	`long create_handle (`
	`  Callbacks callbacks,`
	`  Config config`
	`);`

Descripción	Crea un *handle* para usar en funciones de clase Mjpeg. Se debe realizar una llamada a '*destroy_handle*' para liberar los recursos asignados.
Parámetros	Un objeto que implementa la '*interfaz Mjpeg.Callbacks' y un objeto* Mjpeg.Config *' que contiene los parámetros de configuración del flujo*.
Retorno	En caso de éxito, devuelve la dirección de memoria del *handle* creado. De lo contrario, devuelve '*0*'.

Mjpeg.destroy_*handle*	
Prototipo de Función	`void destroy_handle (long handle);`

Descripción	Libera los recursos asignados al *handle*. Para evitar que un *handle* ya liberado se reutilice incorrectamente, se recomienda que después de llamar a esta función, se asigne *'0'* al valor del *handle*, es decir `mHandle = 0;`.
Parámetros	Un `*long*` que contiene la dirección de memoria de un *handle* Mjpeg válido.
Retorno	'*0* si no se puede crear el *handle*. De lo contrario, devuelve un valor distinto de cero.

Mjpeg.connect	
Prototipo de Función	`void connect (long handle);`
Descripción	Intenta establecer una conexión con la URL definida al crear el *handle* Mjpeg.
Parámetros	Un `*long*` que contiene la dirección de memoria de un *handle* Mjpeg válido.
Retorno	`*JidoshaLight.SUCCESS*` en caso de éxito. `*Mjpeg.JL_MJPEG_CONNECT_FAILED*` si la conexión no se puede establecer inmediatamente. En este caso, el *handle* no intentará reconectarse automáticamente, dejando que el usuario llame a '*connect*' nuevamente en el momento oportuno.

Mjpeg.get_frame	
Prototipo de Función	`byte[] get_frame(long handle)`
Descripción	Devuelve el frame más reciente de la cola de recepción de Mjpeg. Las llamadas consecutivas a esta función pueden devolver el mismo frame si no se han recibido nuevos marcos en el intervalo.
Parámetros	Un `*long*` que contiene la dirección de memoria de un *handle* Mjpeg válido.
Retorno	Un array de bytes que contiene el último frame recibido en formato JPEG. Si no se ha recibido ningún marco en el momento de la llamada, la función devuelve un array de tamaño 0 '*byteArray.length == 0' y la* callback 'error_cb' *con el código* 'JL_LAST_FRAME_UNAVAILABLE'.

6.4 Guía de Migración - API 1 C/C++ JIDOSHA

El proceso de migración de una aplicación de PC que usa API 1 de la biblioteca JIDOSHA a una aplicación embarcada con la biblioteca JidoshaLight es simple y rápido:

1. la función '*lePlaca*' debe ser reemplazada por la función '*jidoshaLight_ANPR_fromFile*';
2. la 'struct *JidoshaConfig*' debe ser reemplazado por 'struct *JidoshaLightConfig*';
3. la 'struct *Recognition*' por 'struct *JidoshaLightRecognition*'.

El usuario debe estar atento a los nuevos campos de configuración de JidoshaLight, los cuales necesariamente deben ser llenados con los valores correctos.

El siguiente ejemplo muestra cómo lograr el mismo comportamiento de JIDOSHA con 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;
}
```

Observaciones:

- La `*struct JidoshaLightConfig config*` se inicializa a cero '*{0}*', asegurando que los campos '*int xRoi[4]* e '*int yRoi[4]* sean cero y deshabilitando el uso del ROI.
- El modo de procesamiento '*JIDOSHA_LIGHT_MODE_ULTRA_SLOW*' es el que más se asemeja a la estrategia de procesamiento utilizada por la biblioteca JIDOSHA.

El SDK viene con una aplicación de ejemplo más detallada.

7 APIs de usuario de JIDOSHA

Para una mayor facilidad de uso y migración a la biblioteca JidoshaLight, las API de JIDOSHA también están disponibles a través de las bibliotecas *'libjidoshaCore.so'* y *'jidoshaCore.dll'*. Es posible intercambiar la biblioteca JIDOSHA por estos nuevos archivos, manteniendo el mismo comportamiento (con algunas advertencias; consulte Builds especiales de la API heredada. Los archivos con la interfaz JIDOSHA se encuentran dentro de la carpeta jidoshapc del SDK de Windows o Linux.

La API (Application Programming Interface) nativa de JIDOSHA está escrita en lenguaje C, lo que permite su uso desde cualquier idioma. El SDK también incluye bibliotecas contenedoras para simplificar el uso de la biblioteca de .NET (C# y VB.NET), Java y Delphi. Estos wrappers simplemente envuelven llamadas a funciones de biblioteca, haciendo cualquier conversión necesaria de parámetros y resultados.

Toda la API de C está disponible a través de un único archivo de header, jidoshaCore.h, cuyo contenido se presenta a continuación. También se proporciona una descripción más detallada.

La biblioteca se puede utilizar de dos formas: a través de API 1 o API 2:

La API 1, que fue la primera API de JIDOSHA, tiene como principal motivación la facilidad de uso. Es posible leer matrículas a través de una sola llamada de función ('*lePlaca*' o '*lePlacaFromMemory*', en el caso del lenguaje C).

La API 2 se creó para brindar una mayor flexibilidad en la configuración de la biblioteca y la carga de imágenes. Por ejemplo, puede configurar el número mínimo de caracteres que deben leerse con buena fiabilidad para que la placa se considere válida. Es posible agregar nuevos parámetros de configuración a la API 2 sin afectar a los usuarios existentes de la biblioteca (es decir, estos usuarios pueden actualizar JIDOSHA DLL/.so a una versión más nueva, sin tener que volver a compilar). Además, API 2 permite el uso de imágenes RAW, tanto en grayscale como RGB/BGR. Se puede agregar compatibilidad con otros formatos según sea necesario.

Recomendamos API 1 para cualquiera que necesite integrar JIDOSHA en su aplicación lo más rápido posible y API 2 para cualquiera que desee un mayor control sobre el funcionamiento de la 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	
Descripción	El objetivo de esta estructura es configurar el comportamiento de la biblioteca en la llamada de reconocimiento de matrículas.
Miembros	*int tipoPlaca*: indica el tipo de placa que debe buscar el OCR, que debe ser uno de los siguientes valores: • JIDOSHA_TIPO_PLACA_CARRO: Solo se buscarán placas, donde "auto" significa "no motocicleta", es decir incluye autos, camionetas, buses, etc. • JIDOSHA_TIPO_PLACA_MOTO: solo se buscarán placas de moto. • JIDOSHA_TIPO_PLACA_AMBOS: Se buscarán matrículas tanto de moto como de no moto. *int timeout*: indica el tiempo máximo que debe tardar el reconocimiento de matrículas, en milisegundos. Un valor de cero indica que no hay tiempo de espera. Un valor distinto de cero ayuda a mantener bajo el tiempo promedio de procesamiento. El valor debe determinarse en función de la resolución de la imagen y la CPU utilizada.

struct Reconhecimento	
Descripción	El propósito de esta estructura es almacenar el resultado del reconocimiento de matrículas, incluyendo: los caracteres de la matrícula, la fiabilidad de cada carácter y las coordenadas de la matrícula en la imagen.
Miembros	*char placa[8]*: placa de 7 caracteres terminada en 0, ou string vacía si no se encuentra la placa. *double probabilities[7]*: valores de 0.0 a 1.0 que indican la fiabilidad, en forma de probabilidad, de reconocer cada carácter.

	int xText e *int yText*: coordenadas del punto superior izquierdo de la placa, si se encuentra. *int widthText*: ancho del rectángulo de la placa. *int heightText*: altura del rectángulo de la placa. *int textColor*: color del texto de la placa, 0 - oscuro, 1 - claro. *int isMotorcycle*: indica si la matrícula es de moto, 0 - no moto, 1 - moto.

7.2.2 Métodos

lePlaca	
Prototipo de Función	`int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);`
Descripción	Reconoce la matrícula y la almacena en un objeto '*Reconocimiento*'. La imagen debe pasarse como parámetro en el formato de path donde se encuentra la imagen. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto '*Reconocimiento*' contendrá una string vacía como matrícula. El archivo de imagen debe ser un mapa de bits, jpeg o png.
Parámetros	*filename*: path al archivo de imagen. *config*: puntero a la struct '*JidoshaConfig*' con la configuración de la biblioteca. *rec*: puntero a la *struct 'Reconocimiento'* donde se almacenará el resultado de la lectura.
Retorno	Código de error: 0 (cero) en caso de éxito, número distinto de cero en caso contrario.

lePlacaFromMemory	
Prototipo de Función	`int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Reconhecimento*rec);`
Descripción	Reconoce la matrícula y la almacena en un objeto '*Reconocimiento*'. La imagen debe pasarse como un parámetro en formato de array de bytes, y el número de bytes indicado por el parámetro '*n*'. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula. El archivo de imagen debe ser un mapa de bits, jpeg o png.
Parámetros	*stream*: array de bytes que contiene la imagen. *n*: tamaño del array de bytes. *config*: puntero a la struct *'JidoshaConfig'* con la configuración de la biblioteca. *rec*: puntero a la *struct 'Reconocimiento'* donde se almacenará el resultado de la lectura.
Retorno	Código de error: 0 (cero) en caso de éxito, número distinto de cero en caso contrario.

getVersion	
Prototipo de Función	`int getVersion(int* major, int* minor, int* release);`
Descripción	Se utiliza para comprobar la versión de la biblioteca, en formato major.minor.release.
Parámetros	*major*: puntero a la variable int donde se escribirá major. *minor*: puntero a la variable int donde se escribirá minor. *release*: puntero a la variable int donde se escribirá el release.

Retorno	Siempre devuelve 0 (cero).

getHardkeySerial	
Prototipo de Función	`int getHardkeySerial(unsigned long* serial);`
Descripción	Se utiliza para comprobar el número de serie de la hardkey.
Parámetros	*serial*: puntero a la variable unsigned long donde se escribirá el número de serie de la hardkey.
Retorno	Devuelve 0 en caso de éxito, 1 si no se encontró la hardkey.

getHardkeyState	
Prototipo de Función	`int getHardkeyState(int* state);`
Descripción	Se utiliza para comprobar el estado de la hardkey. Si el estado es igual a 0, la hardkey no está autorizada; si el estado es igual a 1, la hardkey está autorizada.
Parámetros	*state*: puntero a la variable int se escribirá el estado de la hardkey.
Retorno	Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

getHardkeyRemainingTime	
Prototipo de Función	`int getHardkeyRemainingTime(int* days, int* hours);`
Descripción	Se utiliza para comprobar el tiempo restante de las licencias de demostración. Si day y hour son iguales a -1 no hay límite de tiempo.
Parámetros	*days*: puntero a variable int donde se escribirá el número de días restantes. *hours*: puntero a variable int donde se escribirá el número de horas restantes.
Retorno	Devuelve 0 en caso de éxito, 1 si no se encontró la hardkey.

7.3 API2 JIDOSHA C/C++

7.3.1 Tipos

struct ResultList	
Descripción	El propósito de esta estructura es almacenar la lista enlazada con los resultados del procesamiento de las funciones '*jidoshaFindFirst*' y '*jidoshaFindNext*'.
Miembros	*struct ResultList* next*: puntero al siguiente nodo de la lista. NULL si el nodo actual es el último. *struct Reconocimiento* reconocimiento*: puntero a la *struct que contiene un resultado de reconocimiento de matrícula.*

typedef void JidoshaHandle	
Descripción	Tipo utilizado para representar la memoria asignada para la configuración.

typedef void JidoshaImage	
Descripción	Tipo utilizado para representar la memoria asignada a una imagen.

7.3.2 Métodos

jidoshaFreeResultList	
Prototipo de Función	`void jidoshaFreeResultList(ResultList* list);`
Descripción	Libera la memoria asignada para la lista enlazada de resultados.
Parámetros	*list*: puntero a un struct `*ResultList*`.
Retorno	No tiene retorno.

jidoshaInit	
Prototipo de Función	`Jidosha*Handle** jidoshaInit();`
Descripción	Asigna memoria para la configuración de la biblioteca. En el caso de uso multithread, cada thread debe llamar a '*jidoshaInit*' y usar su propio '*JidoshaHandle*'.
Parámetros	No tiene.
Retorno	Devuelve un puntero a '*JidoshaHandle*' que se usará en llamadas de funciones posteriores.

jidoshaDestroy	
Prototipo de Función	`int jidoshaDestroy(Jidosha*Handle** handle);`
Descripción	Libera la memoria asignada por la función jidoshaInit.
Parámetros	*handle*: puntero a una variable `*JidoshaHandle*`.
Retorno	JIDOSHA_SUCCESS.

jidoshaSetIntProperty	
Prototipo de Función	`int jidoshaSetIntProperty(Jidosha*Handle** handle, const char* name, int value);`
Descripción	Cambia el valor de una variable de tipo int en la configuración.
Parámetros	*handle*: puntero a `Jidosha*Handle*`. *name*: string que contiene el nombre de la propiedad a cambiar. *value*: valor a asignar a la propiedad.

Retorno	JIDOSHA_SUCCESS si se cambia el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo int.

jidoshaGetIntProperty	
Prototipo de Función	`int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);`
Descripción	Lee el valor de una variable de tipo int de la configuración.
Parámetros	*handle*: puntero a `*JidoshaHandle*`. *name*: string que contiene el nombre de la propiedad a leer. *value*: puntero a la variable int donde se escribirá el valor de la propiedad.
Retorno	JIDOSHA_SUCCESS si se cambia el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo int.

jidoshaSetDoubleProperty	
Prototipo de Función	`int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name, double value);`
Descripción	Cambia el valor de una variable de tipo double en la configuración.
Parámetros	*handle*: puntero a `*JidoshaHandle*`. *name*: string que contiene el nombre de la propiedad a cambiar. *value*: valor a asignar a la propiedad.
Retorno	JIDOSHA_SUCCESS si se cambia el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo double.

jidoshaGetDoubleProperty	
Prototipo de Función	`int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name, double* value);`
Descripción	Lee el valor de una variable de tipo double desde la configuración.
Parámetros	*handle*: puntero a `*JidoshaHandle*`. *name*: string que contiene el nombre de la propiedad a leer. *value*: puntero a la variable doble donde se escribirá el valor de la propiedad.
Retorno	JIDOSHA_SUCCESS si se lee el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo double.

jidoshaSetCharProperty	
Prototipo de Función	`int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);`
Descripción	Cambia el valor de una variable de tipo char en la configuración.
Parámetros	*handle*: puntero a `*JidoshaHandle*`.

	name: string que contiene el nombre de la propiedad a cambiar. *value*: valor a asignar a la propiedad.
Retorno	JIDOSHA_SUCCESS si se cambia el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo char.

jidoshaGetCharProperty	
Prototipo de Función	`int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);`
Descripción	Lee el valor de una variable de tipo char de la configuración.
Parámetros	*handle*: puntero a `*JidoshaHandle*`. *name*: string que contiene el nombre de la propiedad a leer. *value*: puntero a la variable char donde se escribirá el valor de la propiedad.
Retorno	JIDOSHA_SUCCESS si se lee el valor de la variable, JIDOSHA_ERROR_INVALID_PROPERTY si la propiedad no existe o no es de tipo char.

jidoshaFindFirst	
Prototipo de Función	`int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);`
Descripción	Reconoce la matrícula y la almacena en un objeto *'Reconocimiento'* que se encuentra en el primer nodo de '*ResultList*'. La imagen debe cargarse mediante las funciones '*jidoshaLoadImage*' o '*jidoshaLoadImageFromMemory*'. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula. Esta función solo debe llamarse con una '*ResultList*' vacía.
Parámetros	*handle*: puntero a un `*JidoshaHandle*` que contiene la configuración de la biblioteca. *image*: puntero a una `*JidoshaImage*` que contiene la imagen a procesar. *list*: puntero a un `*ResultList*` donde se almacenará el resultado del procesamiento.
Retorno	JIDOSHA_SUCCESS si se procesa la imagen, de lo contrario, otro valor de enum `*jidoshaError*`.

jidoshaFindNext	
Prototipo de Función	`int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);`
Descripción	El propósito de esta función es permitir al usuario reconocer múltiples matrículas en la misma imagen. La función reconoce la matrícula y la almacena en un objeto *'Reconocimiento'* que se encuentra en el último nodo de '*ResultList*'. La imagen debe cargarse mediante las funciones '*jidoshaLoadImage*' o '*jidoshaLoadImageFromMemory*'. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula. Esta función solo debe llamarse con una '*ResultList*' previamente procesada por la función '*jidoshaFindFirst*' o '*jidoshaFindNext*'.
Parámetros	*handle*: puntero a un `*JidoshaHandle*` que contiene la configuración de la biblioteca.

	image: puntero a una `*JidoshaImage*` que contiene la imagen a procesar. *list*: puntero a un `*ResultList*` donde se almacenará el resultado del procesamiento.
Retorno	JIDOSHA_SUCCESS si se procesa la imagen, de lo contrario, otro valor de enum `*jidoshaError*`.

jidoshaLoadImage	
Prototipo de Función	`int jidoshaLoadImage(const char* filename, JidoshaImage** img);`
Descripción	Carga una imagen de un archivo y guarda la referencia como JidoshaImage. El archivo de imagen debe ser un mapa de bits, jpeg o png.
Parámetros	*filename*: path al archivo de imagen. *img*: puntero a puntero a la struct '*JidoshaImage*' donde se almacenará la imagen.
Retorno	JIDOSHA_SUCCESS si la imagen se carga correctamente, JIDOSHA_ERROR_FILE_NOT_FOUND si el archivo no se encuentra o no existe, JIDOSHA_ERROR_INVALID_IMAGE ou JIDOSHA_ERROR_INVALID_IMAGE_TYPE en caso de problemas al cargar la imagen.

jidoshaLoadImageFromMemory	
Prototipo de Función	`int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImage** img);`
Descripción	Carga una imagen de un array de bytes y guarda la referencia como '*JidoshaImage*'. La imagen debe estar en algún formato estructurado (bmp, jpg, png, etc.) o raw (Grayscale 8bit, RGB o BGR).
Parámetros	*buf*: *array* de bytes que contiene la imagen. *n*: tamaño del array en bytes. *type*: tipo de imagen: tipos estructurados=0, GRAY8=1, RGB=2, BGR=3. *width*: ancho de la imagen, ignorado si type==0. *height*: altura de la imagen, ignorada si type==0. *img*: puntero a puntero a '*JidoshaImage*' donde se almacenará la imagen.
Retorno	JIDOSHA_SUCCESS si la imagen se carga correctamente, JIDOSHA_ERROR_FILE_NOT_FOUND si el archivo no se encuentra o no existe, JIDOSHA_ERROR_INVALID_IMAGE ou JIDOSHA_ERROR_INVALID_IMAGE_TYPE en caso de problemas al cargar la imagen.

jidoshaFreeImage	
Prototipo de Función	`int jidoshaFreeImage(JidoshaImage** img);`
Descripción	Libera la memoria asignada para almacenar una imagen.
Parámetros	*img*: puntero a puntero a una '*JidoshaImage*' que será desasignada.
Retorno	JIDOSHA_SUCCESS.

jidoshaBuildInfo	
Prototipo de Función	`const char* jidoshaBuildInfo();`
Descripción	Verifica la información de build de la biblioteca y se usa para verificar que la versión que se está ejecutando es la esperada.
Parámetros	No tiene.
Retorno	String constante que tiene 12 o 13 caracteres que representan BuildInfo más un terminador ('\0').

jidoshaNumThreads	
Prototipo de Función	`int jidoshaNumThreads();`
Descripción	Comprueba el número de threads autorizados en la hardkey.
Parámetros	No tiene.
Retorno	Entero que representa cuántos threads están autorizados para ejecutar simultáneamente las funciones de OCR de la biblioteca. Devuelve 1 si no se encuentra la hardkey.

7.3.3 API 2 - Configuración

En esta sección detallamos todos los parámetros de configuración disponibles en la API 2. Válido para la API C, Java, .NET y Python.

Parámetro *tipoPlaca*	
Descripción	Sirve para restringir el tipo de matrícula que se debe reconocer. Es principalmente interesante reducir el tiempo de procesamiento. En particular, cuando '*tipoPlaca=JIDOSHA_TIPO_PLACA_AUTO*', la biblioteca puede utilizar un método de localización de matrículas más rápido. Los valores válidos son:
Nombre	*tipoPlaca*
Tipo	int
Valor default	JIDOSHA_TIPO_PLACA_AMBOS
Otros valores	• JIDOSHA_TIPO_PLACA_CARRO` == 1 • JIDOSHA_TIPO_PLACA_MOTO` == 2 • JIDOSHA_TIPO_PLACA_AMBOS` == 3

Parámetro *timeout*	
Descripción	Después de los milisegundos de '*timeout*' desde el inicio del procesamiento de una imagen, la búsqueda de matrículas finalizará y se devolverá la mejor matrícula encontrada. Si '*timeout*' es cero, no hay tiempo de espera. Se recomienda utilizar un '*timeout*' distinto de cero cuando la aplicación requiere que las imágenes se procesen rápidamente (con baja latencia) o cuando la carga de la CPU es muy alta.

Nombre	*timeout*
Tipo	int
Valor default	0

Parámetro *minNumChars*	
Descripción	Indica el número mínimo de caracteres que debe tener una matrícula. Si la versión en uso de la biblioteca tiene varias sintaxis de matrículas habilitadas (por ejemplo, matrículas de varios países), este parámetro se ignora y se debe usar '*numAllowedBadChars*' en su lugar.
Nombre	*minNumChars*
Tipo	int
Valor default	7

Parámetro *numAllowedBadChars*	
Descripción	Indica el número máximo de caracteres faltantes que puede tener una matrícula. Este parámetro se utiliza cuando se quiere que se devuelvan matrículas parcialmente reconocidas.
Nombre	*numAllowedBadChars*
Tipo	int
Valor default	0

Parámetro *maxNumChars*	
Descripción	Indica el número máximo de caracteres que debe tener una matrícula. Actualmente este parámetro se ignora.
Nombre	*maxNumChars*
Tipo	int
Valor default	7

Parámetro *minCharWidth*	
Descripción	Ancho mínimo que debe tener un carácter, en píxeles.
Nombre	*minCharWidth*
Tipo	int
Valor default	1

Parámetro *avgCharWidth*

Descripción	Ancho promedio esperado de un carácter, en píxeles. Actualmente este parámetro no se utiliza.
Nombre	*avgCharWidth*
Tipo	int
Valor default	1

Parámetro *maxCharWidth*	
Descripción	Ancho máximo que debe tener un carácter, en píxeles.
Nombre	*maxCharWidth*
Tipo	int
Valor default	7

Parámetro *minCharHeight*	
Descripción	Altura mínima que debe tener un carácter, en píxeles.
Nombre	*minCharHeight*
Tipo	int
Valor default	9

Parámetro *avgCharHeight*	
Descripción	Altura media esperada de un carácter, en píxeles. Este parámetro se puede utilizar cuando las placas son muy grandes. Cuando *'avgCharHeight > 30'*, la imagen se reducirá internamente antes de ser procesada. Los límites de tamaño de carácter mínimo y máximo se ajustarán de acuerdo con el factor de escala.
Nombre	*avgCharHeight*
Tipo	int
Valor default	20

Parámetro *maxCharHeight*	
Descripción	Altura máxima de un carácter, en píxeles.
Nombre	*maxCharHeight*
Tipo	int
Valor default	60

Parámetro *ocrModel*	
Descripción	Define el modelo de OCR que se utilizará para el reconocimiento de caracteres. Este parámetro existe para cambiar fácilmente el modelo de OCR a modelos de versiones anteriores de la biblioteca, sin necesidad de volver a compilar la aplicación de usuario o cambiar la biblioteca. No utilice valores diferentes al predeterminado, excepto cuando sea recomendado por el equipo de soporte de Pumatronix Equipamentos Eletrônicos.
Nombre	*ocrModel*
Tipo	int
Valor default	1

Parámetro *checkSyntax*	
Descripción	Cuando *'checkSyntax=1'*, la biblioteca aplica un paso de procesamiento adicional para verificar que los caracteres reconocidos tengan la sintaxis esperada (letra o número), lo que reduce la incidencia de falsos reconocimientos (textos que no son matrículas). Nota: incluso cuando *'checkSyntax=0'*, la biblioteca nunca devolverá un reconocimiento con una sintaxis diferente a la definida. Por ejemplo, para matrículas brasileñas, la matrícula devuelta siempre tendrá 3 letras seguidas de 4 números. Sin embargo, el texto que no es de placa, como *"ESCOLAR"*, podría confundirse con una placa, lo que resultaría en el reconocimiento como *"ESC0148"*. La sintaxis está de acuerdo con una placa brasileña, a pesar de no ser una placa. Usar *'checkSyntax=1'* puede ayudar a descartar falsos reconocimientos como en el ejemplo.
Nombre	*checkSyntax*
Tipo	int
Valor default	1

Parámetro *minPlateAngle*	
Descripción	Ángulo de inclinación mínimo en grados permitido para una placa. Para obtener más detalles, consulte la sección de configuración de la perspectiva de la imagen.
Nombre	*minPlateAngle*
Tipo	double
Valor default	-30.0

Parámetro *maxPlateAngle*	
Descripción	Ángulo máximo de inclinación en grados permitido para una placa. Para obtener más detalles, consulte la sección de configuración de la perspectiva de la imagen.
Nombre	*maxPlateAngle*
Tipo	double

Valor default	30.0

Parámetro *minProbPerCharacter*	
Descripción	Probabilidad mínima (confiabilidad) requerida para reconocer cada carácter. Es extremadamente importante que el OCR funcione correctamente y no se recomienda cambiar la configuración *default*. No obstante, en casos puntuales puede ser interesante ajustarlo. Si '*minProbPerCharacter*' es menor que el predeterminado, el número de matrículas que no se reconocen disminuirá, pero, por otro lado, el número de matrículas con un carácter incorrecto puede aumentar. Si '*minProbPerCharacter*' es mayor que el *default*, la cantidad de matrículas que no se reconocen puede aumentar, pero la cantidad de errores será menor.
Nombre	*minProbPerCharacter*
Tipo	double
Valor default	0.8

Parámetro *excellentProb*	
Descripción	Este parámetro existe para reducir el tiempo promedio de procesamiento. Si todos los caracteres reconocidos tienen una probabilidad mayor o igual a '*excellentProb*', el reconocimiento se considera excelente y se devuelve al usuario inmediatamente sin más procesamiento. De lo contrario, el procesamiento continuará hasta que se cumpla una de las siguientes condiciones: se encuentra un reconocimiento excelente; se alcanza el tiempo de espera; o no hay más pasos de procesamiento para hacer. Los valores más altos de '*excellentProb*' dan como resultado tasas de reconocimiento más altas y tasas de error más bajas (confusión entre caracteres), pero con un tiempo de procesamiento más largo. Los valores más pequeños de '*excellentProb*' dan como resultado tasas de reconocimiento más bajas y tasas de error más altas (confusión entre caracteres), pero con menos tiempo de procesamiento.
Nombre	*excellentProb*
Tipo	double
Valor default	0.95

Parámetro *lowProbabilityChar*	
Descripción	Carácter de sustitución que se utilizará cuando se reconozca un carácter de matrícula con una probabilidad inferior a '*minProbPerCharacter*'. Solo tiene efecto si '*minNumChars*' es menor que '*maxNumChars*'. Por ejemplo, si '*lowProbabilityChar='-'* y 'minNumChars=6', la placa *'"ABC1234"'* se devolverá *como '"A-C1234"'* si la probabilidad del segundo carácter es menor que 'minProbPerCharacter'.
Nombre	*lowProbabilityChar*

Tipo	char
Valor default	'*'

Parámetro *country*	
Descripción	Carácter de sustitución que se utilizará cuando se reconozca un carácter de matrícula con una probabilidad inferior a '*minProbPerCharacter*'. Solo tiene efecto si '*minNumChars*' es menor que '*maxNumChars*'. Por ejemplo, si '*lowProbabilityChar='-'* y 'minNumChars=6', la placa '"*ABC1234*"' se devolverá como '"*A-C1234*"' si la probabilidad del segundo carácter es menor que 'minProbPerCharacter'.
Nombre	*country*
Tipo	int
Valor default	76

7.3.4 API 2 - Configuración de perspectiva de imagen

En general, se recomienda instalar la cámara de captura de matrículas de forma que las matrículas queden alineadas con los ejes horizontal y vertical de la imagen. Sin embargo, en algunas situaciones esto no es posible y termina obteniendo placas inclinadas en relación con los ejes de la imagen, lo que puede dificultar el reconocimiento de las placas. En estos casos, la perspectiva de la placa puede ser informada a la biblioteca. Luego, la biblioteca realizará una corrección de perspectiva para maximizar la tasa de reconocimiento de placas.

En el caso de equipos con varias cámaras, se recomienda crear un API 2 '*handle*' por cámara (a través de la función '*jidoshaInit*') y configurar los parámetros de perspectiva individualmente para cada '*handle*'.

Los parámetros '*avgPlateAngle*', '*avgPlateSlant*' y '*adjustPerspective*' se utilizan para informar la perspectiva de la placa en la imagen (inclinación horizontal y vertical) y corregirla. La pendiente horizontal ('*avgPlateAngle*') y la pendiente vertical ('*avgPlateSlant*') deben medirse en imágenes de instalación típicas.

Además de configurar manualmente la perspectiva, también es posible habilitar algoritmos en la biblioteca que buscan corregir automáticamente la perspectiva. Consulte los parámetros '*autoSlope*' y '*autoSlant*' para obtener más detalles.

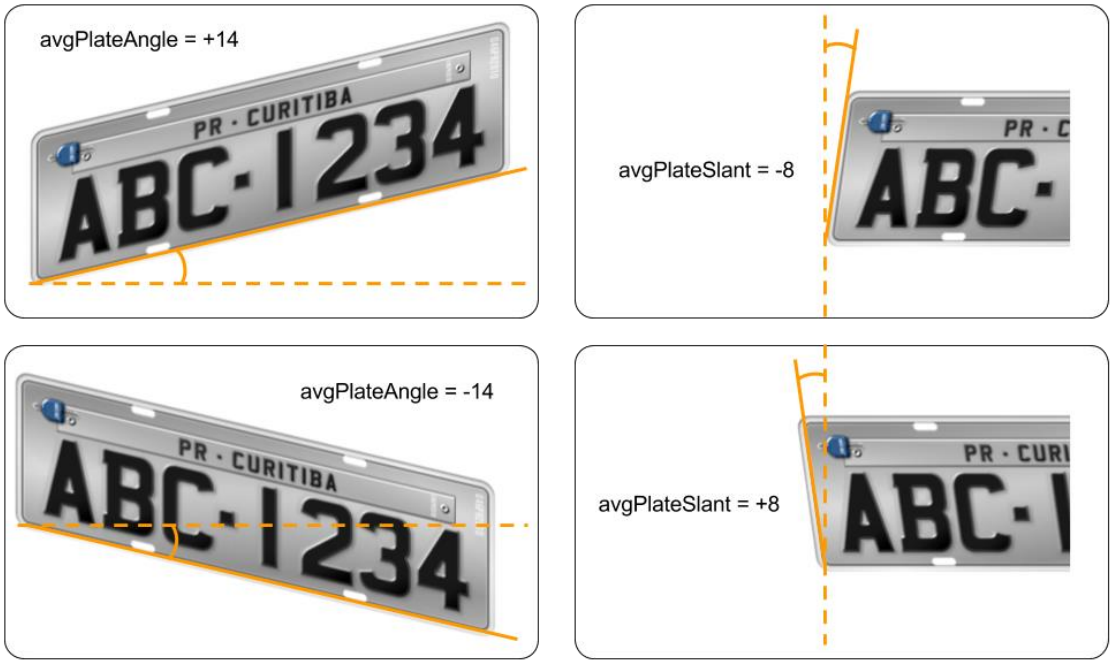


Figura 11 - Cómo calcular los valores avgPlateAngle y avgPlateSlant

Parámetro *avgPlateAngle*	
Descripción	Ángulo de inclinación horizontal promedio en grados esperado para una placa. Se utiliza para realizar el ajuste de la perspectiva de la imagen. Solo tiene efecto si '*adjustPerspective*' es distinto de cero. El ángulo debe medirse según la convención de la imagen anterior.
Nombre	*avgPlateAngle*
Tipo	double
Valor default	0.0

Parámetro *avgPlateSlant*	
Descripción	Ángulo de inclinación vertical promedio en grados esperado para una placa. Se utiliza para realizar el ajuste de la perspectiva de la imagen. Solo tiene efecto si '*adjustPerspective*' es distinto de cero. El ángulo debe medirse según la convención de la imagen anterior.
Nombre	*avgPlateSlant*
Tipo	double
Valor default	0.0

Parámetro *adjustPerspective*	
Descripción	'*adjustPerspective=1*' habilita el conjunto de ajustes de perspectiva a través de '*avgPlateAngle*' y '*avgPlateSlant*'.

	'*adjustPerspective=0*' desactiva el ajuste de perspectiva ('*avgPlateAngle*' y '*avgPlateSlant*' se ignoran).
Nombre	*adjustPerspective*
Tipo	int
Valor default	0

Parámetro *autoSlope*	
Descripción	'*autoSlope=1*' permite el ajuste automático de la pendiente horizontal de la placa. Si se utiliza junto con el ajuste de perspectiva manual ('*avgPlateAngle*' cuando '*adjustPerspective=1*'), el ajuste manual se aplicará antes que el algoritmo de ajuste automático. '*autoSlope=0*' deshabilita el ajuste automático de la pendiente horizontal de la placa.
Nombre	*autoSlope*
Tipo	int
Valor default	1

Parámetro *autoSlant*	
Descripción	'*autoSlant=1*' permite el ajuste automático de la pendiente vertical de la placa. Si se usa junto con el ajuste de perspectiva manual ('*avgPlateSlant*' cuando '*adjustPerspective=1*'), el ajuste manual se aplicará antes que el algoritmo de ajuste automático. '*autoSlant=0*' deshabilita el ajuste automático de la pendiente vertical de la placa.
Nombre	*autoSlant*
Tipo	int
Valor default	1

7.4 API JIDOSHA C# / VB.NET

La API de .NET de la biblioteca presenta tres funciones sobrecargadas, que facilitan el reconocimiento de matrículas de tres fuentes: un *array* de bytes que contiene la imagen codificada (JPG o BMP), un objeto de tipo '*Image*' o un nombre de archivo. Todos necesitan un objeto '*JidoshaConfig*' como parámetro que sirve para configurar el comportamiento de la biblioteca.

7.4.1 API 1

7.4.1.1 Métodos

reconhecePlaca 1	
Prototipo de Función	`Reconocimiento reconhecePlaca(byte[] array, JidoshaConfig config)`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen (JPG, BMP, etc.) debe pasarse como un array de bytes.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto 'Reconocimiento' contendrá una string vacía como matrícula.

reconhecePlaca 2	
Prototipo de Función	`Reconocimiento reconhecePlaca(Image image, JidoshaConfig config)`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen debe pasarse como un parámetro en forma de un objeto '*Image*'.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula.

reconhecePlaca 3	
Prototipo de Función	`Reconocimiento reconhecePlaca(string filename, JidoshaConfig config)`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen debe pasarse como parámetro en el formato de path donde se encuentra la imagen.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula.

getVersionString	
Prototipo de Función	`String getVersionString()`
Descripción	Se utiliza para comprobar la versión de la biblioteca, en formato major.minor.release
Retorno	Devuelve una string formateada con la versión.

getHardkeySerial	
Prototipo de Función	`int getHardkeySerial()`

Descripción	Se utiliza para comprobar el número de serie de la hardkey.
Retorno	Devuelve un int que contiene el número de serie de la hardkey.

getHardkeyState	
Prototipo de Función	`int getHardkeyState()`
Descripción	Se utiliza para comprobar el estado de la hardkey. Si el estado es igual a 0, la hardkey no está autorizada; si el estado es igual a 1, la hardkey está autorizada.
Retorno	Devuelve el estado de la hardkey (0 o 1, como se describe anteriormente).

7.4.2 Ejemplos API JIDOSHA C# / VB.NET

Ejemplo 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"
: "no autorizado");

            if (args.Length < 1)
            {
                Console.WriteLine("uso: jidoshaNETSample imagen");
                Console.WriteLine("Presione Enter para salir");
                Console.ReadLine();
                return;
            }

            // Cargar la imagen
            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;
            Reconocimiento r = Jidosha.reconocePlaca(filename, cfg);
            System.Console.WriteLine("reconocePlaca: {0}", r.placa);
            r = Jidosha.reconocePlaca(array, cfg);
            System.Console.WriteLine("reconocePlacaFromMemory: {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);

            // Cargar una imagen
            IntPtr JidoshaImg = Jidosha.jidoshaLoadImage(array, 0, 0, 0);

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

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

            // Borra la lista de reconocimientos
            Jidosha.jidoshaFreeResultList(resultList);

            // Libera la imagen
            Jidosha.jidoshaFreeImage(JidoshaImg);
```

```
            // Libera el handle de jidosha
            Jidosha.jidoshaDestroy(JidoshaHandle);

            Console.WriteLine("Presione Enter para salir");
            Console.ReadLine();
        }
    }
            }
```

7.4.2.1 Ejemplo 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 Reconocimiento = Jidosha.reconocePlaca(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	
Prototipo de Función	`function reconhecePlaca(filename: String; config: JidoshaConfig) : Reconhecimento;`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen debe pasarse como parámetro en el formato de path donde se encuentra la imagen.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto '*Reconocimiento*' contendrá una string vacía como matrícula.

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

Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen (JPG, BMP, etc.) debe pasarse como un array de bytes.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto '*Reconocimiento*' contendrá una string vacía como matrícula.

7.5.2 Ejemplo API JIDOSHA Delphi

Nota: Este ejemplo es para Delphi 2007. En las versiones más nuevas de Delphi, puede ser necesario convertir la string de ruta del archivo a AnsiString antes de pasarla a la biblioteca C. También puede ser necesario convertir la string de la placa de AnsiString a Unicode.

```
program JidoshaDelphiSample;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  jidoshaDelphi in 'jidoshaDelphi.pas';

var
    filename: String;
    rec: Reconocimiento;
    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 := reconocePlaca(filename, config);
    Writeln('placa: ', rec.placa);
end.
```

7.6 API JIDOSHA Java

7.6.1 API 1

7.6.1.1 Métodos

reconhecePlaca	
Prototipo de Función	`public static native Reconhecimento reconhecePlaca(String filename, JidoshaConfig config);`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. La imagen debe pasarse como parámetro en el formato de path donde se encuentra la imagen.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula.

reconhecePlacaFromMemory	
Prototipo de Función	`public static native Reconhecimento reconhecePlacaFromMemory(byte[] buf, JidoshaConfig config);`
Descripción	Devuelve un objeto '*Reconocimiento*' que representa el resultado del reconocimiento de matrículas. Este objeto contiene la string de matrículas y la probabilidad (fiabilidad) de cada carácter reconocido. La imagen debe pasarse como un parámetro en formato de *array* de bytes.
Retorno	Objeto de '*Reconocimiento*' que contiene la string que representa la placa, un array de dobles que contienen las probabilidades de los caracteres, las coordenadas del texto de la placa, el color del texto (oscuro o claro) y un campo que indica si la placa es de moto. Si no se encuentra ninguna matrícula, o si la hardkey no está autorizada o no se encontró, el objeto *'Reconocimiento'* contendrá una string vacía como matrícula.

7.6.2 Ejemplo 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 especiales de API heredadas

Por diversas razones, la biblioteca JIDOSHA tenía diferentes tipos de builds para el mismo número de versión, que en general no son compatibles entre sí. La build se puede verificar devolviendo la función '*jidoshaBuildInfo*'. La string de buildInfo tiene el siguiente formato: "hash_build", donde "hash" es el hash de la confirmación y "build" es una string que indica el tipo de build.

Hasta la versión 3.4.0, JidoshaLight solo es compatible con la build '*std*' de JIDOSHA, que es la *b*uild predeterminada. A partir de la versión 3.5.0, JidoshaLight también es compatible con la build '*charpos*' ("character positions"), siempre que haya una clave de registro o una variable de entorno, como se detalla a continuación. La única diferencia entre la versión '*std*' y la versión '*charpos*' consiste en cuatro campos adicionales en la estructura '*Reconocimiento*' en el header `*jidoshaCore.h*`, que contienen las coordenadas de los caracteres de la matrícula cuando el reconocimiento es exitoso. Esta diferencia en la API hace que las builds '*std*' y '*charpos*' sean incompatibles (un ejecutable compilado para una de estas builds no se puede usar con la otra).

Nota: El modo de compatibilidad para la build 'charpos' solo se admite en la API de lenguaje C.

Como referencia, las estructuras de las builds '*std*' y '*charpos*' se enumeran a continuación.

7.7.1 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;
```

7.7.2 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 activar el modo de compatibilidad con el build '*charpos*' en **Windows**, es necesario crear una clave en el registro de Windows, en '*HKLM|SOFTWARE|PUMATRONIX*', con nombre '*JL_LEGACY_API_TYPE*', tipo '*REG_SZ*', y valor '*charpos*'. Cualquier otro valor hará que JidoshaLight vuelva a su comportamiento predeterminado (compatibilidad con la build '*std*'). En lugar de registrarse, puede utilizar una variable de entorno, con nombre '*JL_LEGACY_API_TYPE*' y valor '*charpos*'.

La clave de registro se puede crear con el siguiente comando en el indicador (se requieren credenciales de administrador):

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

Para desactivar el modo de compatibilidad con la build '*charpos*', cambie el valor de la variable a una string vacía, o simplemente elimine la clave:

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

Para activar el modo de compatibilidad con la build '*charpos*' en **Linux**, es necesario crear una variable de entorno, con nombre '*JL_LEGACY_API_TYPE*' y valor '*charpos*'. Cualquier otro valor hará que JidoshaLight vuelva a su comportamiento predeterminado (compatibilidad con la build '*std*').

Observaciones:

- Si el modo de compatibilidad de build '*charpos*' está habilitado (*'JL_LEGACY_API_TYPE= charpos'*), pero el código de usuario está utilizando erróneamente la estructura '*std*' de build '*Reconocimiento*', puede ocurrir acceso no válido a la memoria o corrupción de datos silenciosos.
- Se recomienda migrar a la API de JidoshaLight lo antes posible.

PUMATRONIX

www.pumatronix.com