Claudio Francese

Customizing a Python driver for IDS cameras

under Raspberry Pi

T.R. 16/2015   August 2015

Abstract

IDS Cameras (www.ids-imaging.com) are widely used devices both in industrial and scientific applications for their high resolution, framerate and expansion capabilities. Other features include the synchronization of the image capture to an external trigger and turning on an external flash during the frame acquisition.

FIGURE 1 - IDS CAMERAS

The software and the drivers provided by the manufacturer are compatible with Windows ™, Linux for Intel™ architecture and some Linux based embedded systems. One of the supported platforms is the Raspberry Pi (www.raspberrypi.org).

FIGURE 2 - RASPBERRY PI 2 MODEL B

The Raspberry Pi embedded system is a small dimensions computer (85.6mm x 56mm x 21mm) running a Linux operating system on a 900 MHz quad core CPU. The presence of USB and Ethernet connections and an expansion connector (with SPI, I2C signals available) makes the device a good and low cost solution (less than 50 €) for setting up a scalable, reliable network of distributed intelligent controllers for electronic devices, digital integrated circuits (for example DDS signal generators, FPGAs, A/D and D/A converters) and measurement instrumentation.

This document shows how to interface a USB IDS camera to a Raspberry Pi using the Python language and how to modify a working Python wrapper to the IDS camera C library in order to fit the user's needs.

The official IDS Camera driver and SDK for Raspberry Pi

At the time of writing, the driver and SDK for IDS Cameras can be downloaded for free from the manufacturer's website at the address https://en.ids-imaging.com/download-ueye-emb-hardfloat.html

The provided package consists of a gzipped tarball which has to be extracted in the root directory / of the Raspberry Pi filesystem.

```
# tar xvf uEyeSDK-[version number]-ARM_LINUX_IDS_[setup type].tar -C /
```

The software is then installed by means of the command

```
# /usr/local/share/ueye/bin/ueyesdk-setup.sh
```

Interfacing the IDS driver to the Python language

Interfacing the C-style calls to the IDS driver to the Python-style calls of the user program is done by the Python wrapper which can be downloaded from the *NC State Aerial Robotics Club* GitHub repository at the address https://github.com/ncsuarc/ids

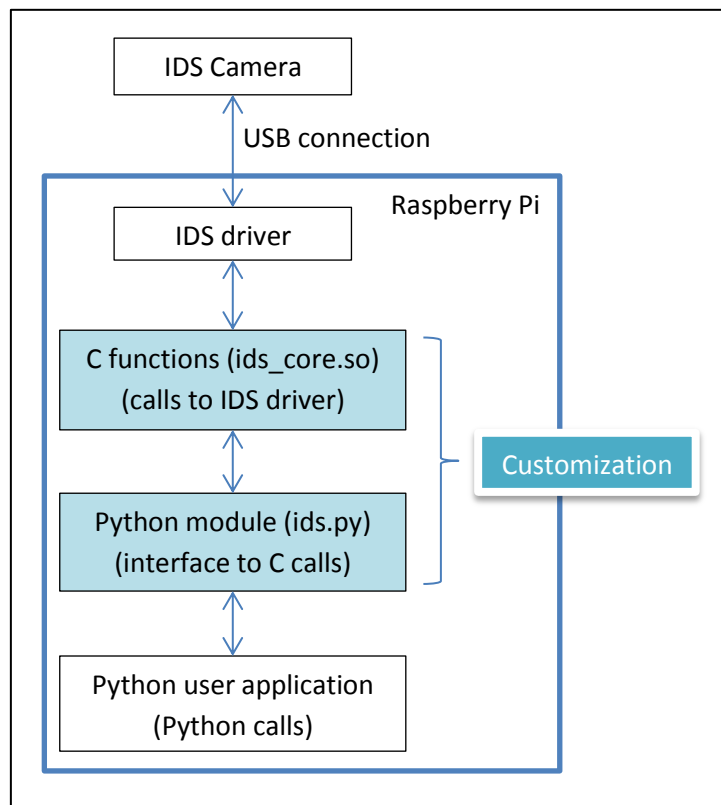A model of the software layers calls is given in Figure 3.

FIGURE 3 – MODEL OF THE SOFTWARE LAYERS

The project has been discontinued in 2013 so the implementation of the interface to the IDS C functions is only partial, thus the wrapper has to be customized according to the users's needs.

The IDS wrapper is installed in the Python module path by means of the command

```
# python setup.py install
```

When the setup command is issued, the sources contained in the `ids_core` directory are compiled

```
ids.py
ids_core | ids_core_constants.c
         | ids_core.h
         | ids_core_methods.c
         | ids_core_Camera_methods.c
         | ids_core.c
         | ids_core_Camera.c
         | ids_core_color.c
         | ids_core_Camera_attributes.c
```

resulting in the `build/lib.linux-armv7l-2.7/ids_core.so` shared object and a Python file `ids.py`.

Properties and methods

The IDS module actually is a wrapper which acts as an interface between the Python language and the IDS driver. Calling a camera object method (see Table 1) or reading/writing a camera property (see Table 2) in Python, turns out into a call to the IDS driver (written in C). The relevant files for the user are the Python file named *ids.py* and the shared object (.so) named *ids_core.so* which is built during the library installation.

Method	Description
capture_status	Get internal camera and driver errors
alloc	Allocates a single memory location for storing images
free_all	Frees all allocated memory for storing images
close	Closes open camera
next_save	Saves next available image
next	Gets the next available image from the camera as a Numpy array

TABLE 1 - CAMERA OBJECT METHODS

Property name	Description
info	Camera info
name	Camera manufacturer and name
width	Image width
height	Image height
pixelclock	Pixel Clock of camera
color_mode	Color mode of images
gain	Hardware gain (individual RGB gains not yet supported)
exposure	Exposure time
auto_exposure	Auto exposure
auto_exposure_brightness	Auto exposure reference brightness (0 to 1)
auto_speed	Auto speed
auto_white_balance	Auto White Balance
color_correction	IR color correction factor
continuous_capture	Enable or disable camera continuous capture (free-run) mode

TABLE 2 - CAMERA OBJECT PROPERTIES

The code snippet in Figure 4 shows how the library can be used.

```
import ids
cam = ids.Camera()
cam.color_mode = ids.ids_core.COLOR_RGB8     # Get images in RGB format
cam.exposure = 5                             # Set initial exposure to 5ms
cam.auto_exposure = True
cam.continuous_capture = True                # Start image capture

        while True:
        img, meta = cam.next()               # Get image as a Numpy array
        process(img)                         #  Process acquired frame
```

FIGURE 4 - PYTHON ID CAMERA MODULE EXAMPLE

The simple API of the IDS wrapper, and the ease of use of the library allow the user to focus on the solution of his/her research problem forgetting the technical details of the camera communication. This encourages the author of this report to extend the capabilities of the wrapper thus hiding to the user many useless programming details.

Extending the IDS C wrapper capabilities

Given the sources of the Python wrapper, the module capabilities can be extended by adding new properties and methods to the *camera* object.

Changing the camera resolution

Among the properties given in Table 2, the IDS python module defines the two properties *width* and *height* which are supposed to be used to change the resolution of the acquired image.

Unfortunately the actual implementation is missing as the authors of the original library did not code it as shown in the source in Figure 5.

```
…
static int ids_core_Camera_setheight(ids_core_Camera *self, PyObject *value, void *closure) {
    PyErr_SetString(PyExc_NotImplementedError, "Changing image height not yet supported.");
    return -1;
    }
…
static int ids_core_Camera_setwidth(ids_core_Camera *self, PyObject *value, void *closure) {
    PyErr_SetString(PyExc_NotImplementedError, "Changing image width not yet supported.");
    return -1;
    }
…
PyGetSetDef ids_core_Camera_getseters[] = {
    …
    {"width", (getter) ids_core_Camera_getwidth,
    (setter) ids_core_Camera_setwidth, "Image width", NULL},
        {"height", (getter) ids_core_Camera_getheight,
    (setter) ids_core_Camera_setheight, "Image height", NULL},
    …
};
```

FIGURE 5 - UNIMPLEMENTED ATTRIBUTES IN IDS_CORE_CAMERA_ATTRIBUTES.C

This way, calling the setter of either the property *width* or *height* has no effect but an error message.

Browsing the IDS SDK documentation at
https://en.ids-imaging.com/manuals/uEye_SDK/EN/uEye_Manual/index.html (login required)
shows that a direct call to a function which sets only one of the two parameters is missing but the function

```
INT is_ImageFormat (HIDS hCam, UINT nCommand, void *pParam, UINT nSizeOfParam)
```

which sets both width and height could be used instead.

A set of 30 resolutions configurations is supported by our camera (UI-149x)

Code	Width	Height		Code	Width	Height
1	3264	2448		16	352	288
2	3264	2176		17	288	352
3	3264	1836		18	320	240
4	2592	1944		19	240	320
5	2048	1536		20	1600	1200
6	1920	1080		21	3840	2748
7	1632	1224		22	1920	1080
8	1280	960		23	2560	1920
9	1280	720		24	768	576
10	–	–		25	1280	1024
11	960	480		26	2448	2048
12	800	480		27	1024	768
13	640	480		28	1024	1024
14	640	360		29	800	600
15	400	240		30	1360	1024

TABLE 3 - CAMERA RESOLUTIONS

where the code of the resolution is stored in the pParam element which is passed to the function. This suggests to add a new property named *imageformat* to the *Camera* object defined in the IDS wrapper.

It should be noted that changing the resolution has the side effect of changing also the size of the buffer needed to store the current grabbed frame, for this reason the buffer needs to be reallocated.

It should be noted that a property with a similar side effect already exists in the wrapper, the *colormode* property. When the color mode is changed also the size of the image changes accordingly.

For this reason, the implementation of the *colormode* property will be used as a starting point to code the *imageformat* property. For this reason the source code for both the functions will be similar.

Implementation of the C function in ids_core_Camera_attributes.c

The interface of the attribute *imageformat* to the manufacturer's driver must be implemented in the C file *ids_core_Camera_attributes.c*

When the user program accesses the attribute *imageformat* in the Python language, the *ids.py* module actually passes the arguments to or gets the results from some functions in *ids_core_Camera_attributes.c*

What is needed is the implementation of the functions which will be called when the property is accessed in read or write mode. This is done by defining the setter and getter functions in the *PyGetSetDef* object at the end of the C file.

In this case the property *imageformat* will have the two getter/setter functions associated

```
PyGetSetDef ids_core_Camera_getseters[] = {
        {"imageformat",        (getter) ids_core_Camera_getimageformat,
                               (setter) ids_core_Camera_setimageformat,
                               "Change the image format", NULL},
              …
        };
```

The 1st string in the dictionary is the name of the property, the 2nd and 3rd elements are the getter and setter functions of the property and the 4th element is the help string which will be shown to the user when requested.

Figure 6 shows the implementation of the attribute setter and getter functions.

```
…

static int ids_core_Camera_setimageformat(ids_core_Camera *self, PyObject *value, void *closure)
{
        int formatcode = (int) PyLong_AsLong(value);
        int nRet;
        UINT count;
        UINT bytesNeeded = sizeof(IMAGE_FORMAT_LIST);
        nRet = is_ImageFormat(self->handle, IMGFRMT_CMD_GET_NUM_ENTRIES, &count, 4);
        bytesNeeded += (count - 1) * sizeof(IMAGE_FORMAT_INFO);
        void* ptr = malloc(bytesNeeded);

        // Create and fill list
        IMAGE_FORMAT_LIST* pformatList = (IMAGE_FORMAT_LIST*) ptr;
        pformatList->nSizeOfListEntry = sizeof(IMAGE_FORMAT_INFO);
        pformatList->nNumListElements = count;
        nRet = is_ImageFormat(self->handle, IMGFRMT_CMD_GET_LIST, pformatList, bytesNeeded);

        IMAGE_FORMAT_INFO formatInfo;
        formatInfo = pformatList->FormatInfo[formatcode];
        nRet = is_ImageFormat(self->handle, IMGFRMT_CMD_SET_FORMAT, &formatInfo.nFormatID, 4);

        self->width = formatInfo.nWidth;
        self->height = formatInfo.nHeight;
        return nRet;
}
…
PyGetSetDef ids_core_Camera_getseters[] = { …
```

FIGURE 6 - IMPLEMENTATION OF THE ATTRIBUTE SETTER AND GETTER FUNCTIONS

Implementation of the Python function in ids.py

The implementation of the setter of the *imageformat* property is very similar to the *colormode* one as at the end of the function the buffer must be reallocated.

Thus when the property is written, not only the corresponding setter function will be called but the memory reallocation functions have to be called. This is done by overriding the property handler, adding the explicit call to the setter and then calling the self.free_all() and self._allocate_memory() functions.

Figure 7 shows the overridden properties accessors for color_mode and imageformat

```
…


# Override color_mode to reallocate memory when changed
@property
    def color_mode(self):
        return ids_core.Camera.color_mode.__get__(self)

    @color_mode.setter
    def color_mode(self, val):
        if self.continuous_capture:
            raise IOError("Color cannot be changed while capturing images")
        ids_core.Camera.color_mode.__set__(self, val)
        # Free all memory and reallocate, as bitdepth may have changed
        self.free_all()
        self._allocate_memory()


…


  # Override imageformat to reallocate memory when changed
    @property
    def imageformat(self):
        return ids_core.Camera.imageformat.__get__(self)

    @imageformat.setter
    def imageformat(self, val):
        if self.continuous_capture:
            raise IOError("Image format cannot be changed while capturing images")
        ids_core.Camera.imageformat.__set__(self, val)
        # Free all memory and reallocate, as resolution has changed
        self.free_all()
        self._allocate_memory()
```

FIGURE 7 - PROPERTIES IN IDS.PY

Code example and results

The resolution can be changed on the fly by calling the following code

```
camera = ids.Camera()

…
camera.continuous_capture = False          # Stop image capture
camera.imageformat         = code
camera.continuous_capture = True           # Restart image capture
…
```
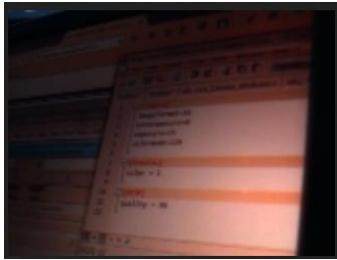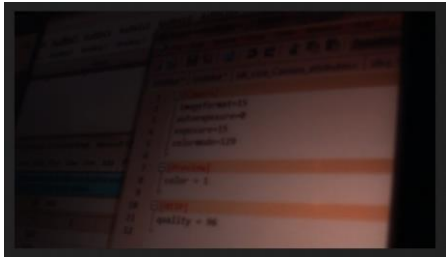
Python code	Image result

```camera.imageformat = 18```
 	
 ```# 320 pixels X 240 pixels```	

```camera.imageformat = 15```
 	
 ```# 400 pixels X 240 pixels```	

TABLE 4- CODE EXAMPLES AND RESULTS

Online resources

www.ids-imaging.com   IDS Imaging Development Systems GmbH

art1.mae.ncsu.edu   NC State Aerial Robotics Club

www.raspberrypi.org   Raspberry Pi Foundation

www.python.org   Python Software Foundation

Appendix – Patching and building the source files

Only two files of the Python wrapper have been modified with respect to the original version (Release date 13 dec 2013) so modifying the wrapper as described in this report can be done by patching the sources of the two files *ids.py* and *ids_core/ids_core_Camera_attributes.c*

In order to patch the files and install the modified python wrapper, please download and uncompress the IDS wrapper from https://github.com/ncsuarc/ids/archive/master.zip

Enter the directory of the uncompressed files then copy the following text [1] and then paste and execute it in a bash session in that directory (for example */home/pi/ids-master* on a Raspberry system)

```
cat <<ENDOFFILE | base64 -d | gunzip | patch -r ids.py
H4sIAACwxFUAA41TW2vCMBR+91cc6osOLYhONhlDkA2EQcE/ULL0tA3k4k5Sh/9+iW01isjydpJz
+S4ncz4fvMFwMJ1OB+8whMFswWeLm6vl82T28lrMluGhPxZlmZaEmDMpR+Pbl3BrOHOYK1SGjqeM
LmkI2QGJRIEgFKuwNKSYA2eAsC+Dtgx+a9TAa6YrLLry9Z7MHskdu7jAMu4zCvPHqwhQkiRRtDHS
ECjjpzcWC/BVwNneNSR01Tay6QVsOJlGMH5IYXNuCNNN9pXt8ifgRlvHtLMpwDYUnlELKeEbIz5F
1I6VDqklFUbyM6AJMAuuPrch/GkEoUI/AhQ7dkLcoNsxYdHGhMPZZh9EhlYxX860Ni4A6xT18gqJ
d+nfF4/QJ+pICaaQWBrJn+Z5hS7PWxsuSNdxjkXnJXjg3wQOTF6ZKMp2sbzmTujGNB7BCTbeMqeg
R89/lLTGdDv2XwWS8bXGjwnbM+EW93XtED79NwG/Cb2vTBfRapxcJ7RGNk4YDbUP+43/A6joKTeg
AwAA
ENDOFFILE

cd ids_core
cat <<ENDOFFILE | base64 -d | gunzip | patch -r ids_core_Camera_attributes.c
H4sIAOCxxFUAA7VU227aQBB9Dl8x4qEy1CQkqbgoKVJEgFgCE3FRH6pq5dgDuLJ30e6Slkb5987a
BhtIpCpSQZa8u2fOOTPemav6tX9Vvy7dgvk9bntSsinqqZYhX1q0/u0zV2gnXkcYI9cYEEJIG8rd
lceXBIIw9pYIKwyXKw1caNiiBrVZr4Uk+Dk8osQ1/cuVm1KtVit1/puUUWg22naz3QqajRYlpbSn
Q5/Exk8/0ddQDQPFfCGRdb0YpceWqBPShZCxp62jY6gqjBY2PIswgKofCbWRWIGXrFwS9UZyYh8K
vmR9KWLzYl22Gy1ycguv9OQmQq7hWF79k3xu/9mLNvimnzNDn9L4IkD4ChbtVHbm7lRiLYlPvJ3t
YvgEdbIxd9wZ+GLDC8unrUblIgYYEKMK/6BYWM7obtBj/fFkdDdjQ2c6SwkNEYFCxRyTUj9NyWRQ
69AHDCIy7owG/cloxrqjezbozZg7H7GeO5s4vakNnxJxG76kfEXtz5ROcgo1uKxA9U0rjtsfp6Gm
PlVYa0l+Yi+KhG8V2FKM+TJnFxfQlehpBDIIizCKIAqVNkcnWRJhWt8hIUx9TxEVo5mwF6C1Dp+S
2/HCLHpcy+07pcz9H0a7mzgJTdtCUXT+lT5SdGPVLiZjw3F1TgpgvGXXy+ELkWjnS3JwYLm/P/me
X8kfHzI8JcOpB7ogueI5zzTu9/fFPCnPrzDQKxIpwr+ZvZsckw2RQ9BDspmgst7edUfWyqVWvU6j
pW0QL+VC65ZtsGiWaJSVkyY/nDEEVO8A1REwHXwIeoXZ6NuLufPh8NUu/QV8YnYdvwUAAA==
ENDOFFILE

cd ..
sudo python setup.py install
```

---

[1] For typographical and space reasons the patch code has been compressed and coded into ASCII before being added to this report. The commands below decode the data and apply the patch to the original sources