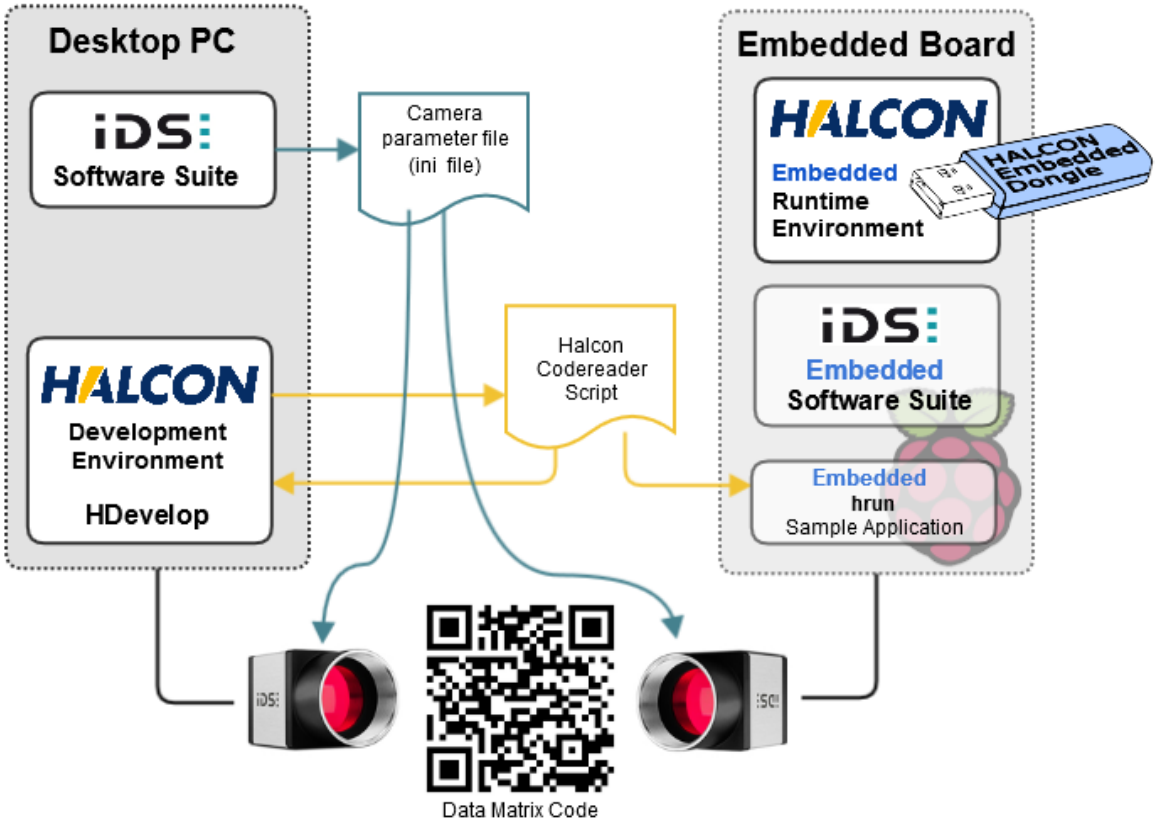
Rapid Prototyping with HALCON Embedded - Code Reading with Raspberry Pi and IDS camera

Just quickly test an embedded image processing application? This task does not have to fail upon the complex cross compiling of source code. Fast and easy setup proof of concepts on an embedded board like the Raspberry Pi with existing components of the HALCON Embedded installation and the IDS embedded driver.

Reading codes like QR, Atztec, ECC200 is implemented very easily with HALCON. HDevelop (HALCON development environment) contains several sample scripts. But how do you run this application on an embedded board? And has the embedded CPU enough power? This application note will give you step-by-step instructions building a simple embedded code reader with already existing components and comparing the performance with a desktop PC.



Using the HALCON script engine (**HDevEngine**) has the benefit that created scripts will run on all supported platforms. The image processing will remain platform independent and therefore expandable via HDevelop. HALCON scripts are created with HDevelop on the development PC and executed with the existing tool **hrun** on the embedded board. With this approach you do not have to cross compile any single line of source code on your own.

Requirements

With a few preparations, this and any other HALCON Embedded application could be realized very fast on a Raspberry Pi 3. You need the following items:

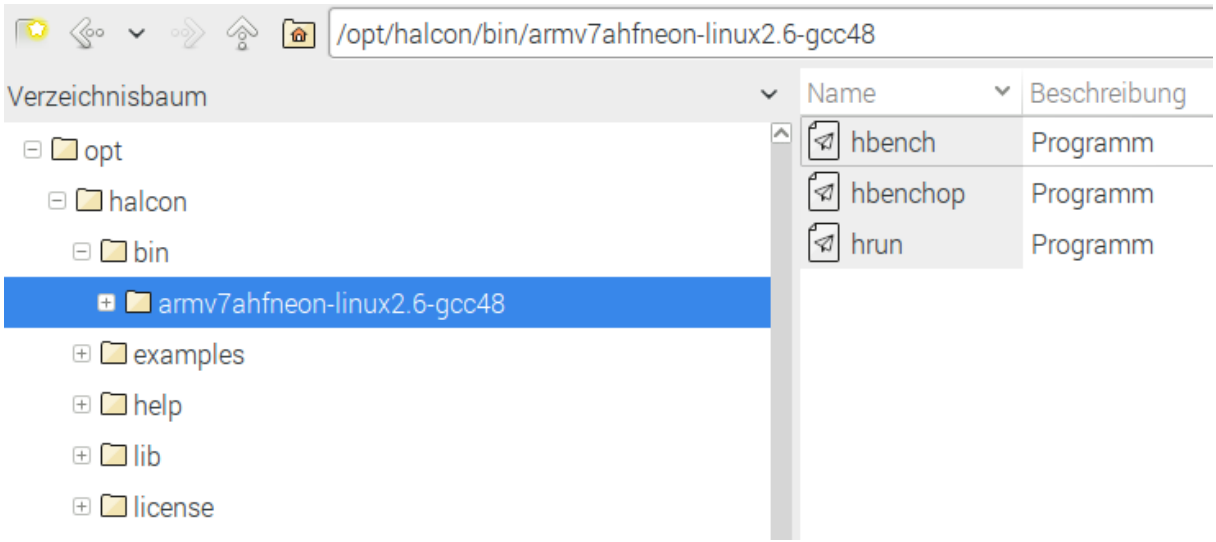
- Raspberry Pi (from version 2, ARMv7 platform compatible), ready installed with Raspbian image and the HALCON Embedded runtime environment.
- Desktop PC with installed (licensed) HALCON development environment
- For using an IDS uEye camera, the IDS embedded driver have to be installed on the Raspberry Pi (Install the embedded driver for IDS industrial cameras)

To try out HALCON Embedded, you can get an evaluation package containing the USB dongle license from IDS or MVTec.

Installing HALCON Embedded

Copy and unpack the HALCON Embedded “Runtime” package on your target platform. This installation path will be your HALCONROOT. (e. g. in `/opt/halcon`)

```
> sudo tar -xvf <paket> -C /opt/halcon
```



To use HALCON for applications in your system, you have to define a few environment variables. Create a shell script, set the appropriate variables (see markings) and save this script e. g. in your HALCONROOT folder. (`/opt/halcon/.profile_halcon`)

```
# Sample shell script for <span class="sp-highlight-term themedarkblue
# sp-cursor-pointer">HALCON</span> environment settings
# (sh syntax)
# If you are using the Bourne shell source this file with the following
# command:
#  source .profile_halcon
HALCONARCH=armv7ahfneon-linux2.6-gcc48; export HALCONARCH
HALCONROOT=/opt/halcon; export HALCONROOT
HALCONEXAMPLES=${HALCONROOT}/examples; export HALCONEXAMPLES
HALCONIMAGES=${HALCONROOT}/examples/images; export HALCONIMAGES
PATH=${HALCONROOT}/bin/${HALCONARCH}:${PATH}; export PATH
if [ ${LD_LIBRARY_PATH} ] ; then
  LD_LIBRARY_PATH=${HALCONROOT}/lib/${HALCONARCH}:${LD_LIBRARY_PATH}; export
LD_LIBRARY_PATH
else
  LD_LIBRARY_PATH=${HALCONROOT}/lib/${HALCONARCH}; export LD_LIBRARY_PATH
fi
```

To activate the new HALCON environment at logon, add the following command to the end of your profile file in your user folder: `~/.profile`

```
source /opt/halcon/.profile_halcon
```

Licensing HALCON Embedded

HALCON Embedded is activated via a license file with the corresponding USB dongle. You can request this license pair from the IDS Support or MVTec. You can look up the license procedure in the */opt/halcon/readme_embedded.txt* file under topic “Licensing”.

- Copy the *license.dat* file, compatible with the USB dongle, in the `/opt/halcon/license` folder on the target system.
- Plug the USB dongle into a free USB port of the Raspberry Pi and check with the **dmesg** command on the Linux console if the dongle is recognized as VendorID 1547.

```
[184728.639356] usb 1-1.4: new low-speed USB device number 5 using dwc_otg
[184728.748339] usb 1-1.4: New USB device found, idVendor=1547, idProduct=1000
[184728.748363] usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[184728.748376] usb 1-1.4: Product: SG-Lock USB Key
[184728.748389] usb 1-1.4: Manufacturer: SG-Lock
[184728.758162] hid-generic 0003:1547:1000.0002: hiddev0,hidraw0: USB HID v1.00 Device
.usb-1.4/input0
```

- Afterwards, create a UDEV rule in the Linux system to allow access to the dongle for all processes. Create the **/etc/udev/rules.d/98-sglock.rules** file with following content:

```
ATTRS{idVendor}=="1547", ATTRS{idProduct}=="1000", MODE="666"
```

- It is necessary to restart the UDEV daemon to activate the new rule. To do so, restart the Raspberry Pi or execute the following command to reload this service:

```
> sudo /etc/init.d/udev reload
```

- Use the **hbench** tool to test the HALCON installation. If error 2036 occurs, this indicates a problem with the license. In this case, check again all steps of the license procedure.

HALCON Benchmark tool

With the **hbench** tool, HALCON provides its own benchmark tool which can be used to evaluate and rate procedure calls on the test system. It is included in each HALCON installation for the corresponding platform (`<HALCONROOT>/bin/<HALCONARCH>/hbench`).

```
pi@raspberrypi:~ $ $HALCONROOT/bin/$HALCONARCH/hbench -operator
find_data_code_2d -reentrant
HALCON 12.0 Benchmark (v3.1)
============================
(computing on images of size 640x480)

OPERATOR                                reentrant
                                         time[ms]
Data code reading, ecc 200 (byte) ......    25.481
```

Figure 1 - HALCON benchmark for operator find_data_code_2d on Raspberry Pi 3

```
C:\Program Files\MVTec\HALCON-12.0\bin\x64-win64\hbench.exe -operator
find_data_code_2d -reentrant
HALCON 12.0.2 Benchmark (v3.1)
==============================
(computing on images of size 640x480)

OPERATOR                                reentrant
                                         time[ms]
Data code reading, ecc 200 (byte) ......     4.811
```

Figure 2 - HALCON benchmark for operator find_data_code_2d on Core i7 Windows PC

The benchmark of the HALCON operator for reading 2D codes (ECC200) shows significant differences in processing time on different platforms.

Simple image based code reading script

With a simple **QR-Code** HALCON script, that reads sample images and is executed on both systems with the **HDevEngine,** you can verify the differences in **reading performance.**

For example, start with the HALCON example script **qrcode_simple.hdev.** You find it in the `/examples/hdevelop/Applications/Data_Codes/` folder. Enhance it with a simple time measurement for the reading process and an output message.

```
* Step 2: Read the data codes
* ---------------------------------------------------
* Search and read the data codes in each image and
* display the decoded string for each found data code
for Index := 1 to ImageNum by 1
    read_image (Image, ImageFiles + Index$'.2d')
    count_seconds(Start)
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], ResultHandles, DecodedDataStrings)
    count_seconds(End)
    Seconds := End - Start
    *
    * Display the results
    dev_display (Image)
    dev_display (SymbolXLDs)
    disp_message (WindowHandle, 'Image ' + Index + ' of ' + ImageNum, 'window', 12, 12, 'black', 'true')
    disp_message (WindowHandle, DecodedDataStrings, 'window', 40, 12, 'black', 'true')
    disp_message (WindowHandle, 'Zeit: '+(Seconds*1000) $ '.2f'+' msec', 'window', 68, 12, 'black', 'true')
```

After testing the script with HDevelop, copy it to the embedded board and execute it directly with the hrun tool. This is delivered pre-compiled with HALCON embedded. You find it in the `<HALCON-ROOT>/bin/<HALCONARCH>/hRun` folder.

```
pi@raspberrypi:~ $ hrun qrcode_simple.hdev
```

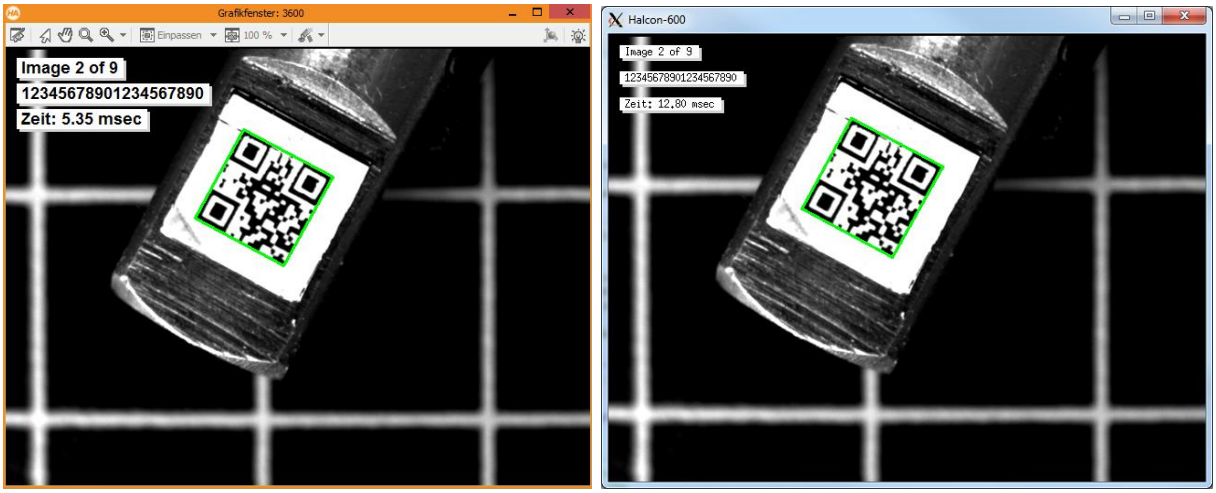


Figure 3 – The time measurement verifies the performance difference of the two systems. (left side: Windows Desktop Core i7, right side: Raspberry Pi 3)

Reading codes with an IDS camera

Next “scan” 2D codes with an IDS uEye camera. Therefore, the IDS Embedded driver has to be installed on the Raspberry Pi. (Embedded boards driver for IDS industrial cameras)

Start with the simple script *“datcode.hdev”,* that is also provided with the HALCON Embedded installation. You find it in the installation part for the development PC.

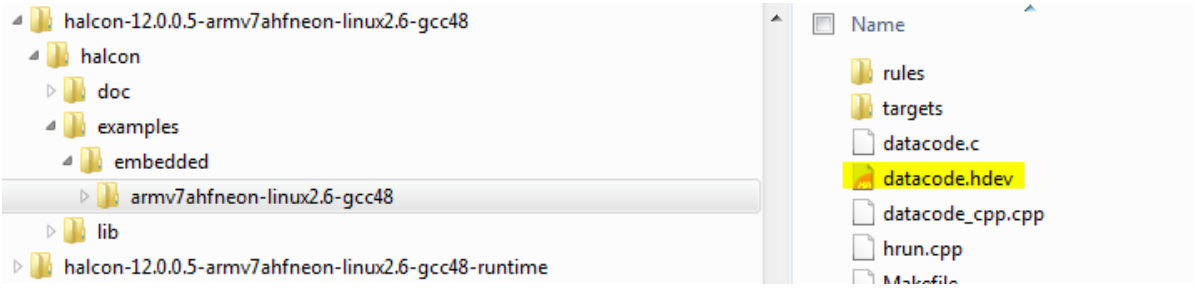


Figure 4 – The HALCON Embedded installation contains the script "datacode.hdev" for reading 2D codes

For using a uEye camera, you only have to modify a few lines of script code. You just need one single "framegrabber" call for all platforms, because the camera's software interface stays the same on all platforms.

```
* get_system ('halcon_arch', HalconArch)
* if (HalconArch = 'armv7ahfneon-linux2.6-gcc46' or HalconArch = 'armv7aneon-linux2.6-gcc46' or HalconArch = 'armv7aneo
*     open_framegrabber ('Video4Linux2', 640, 480, 0, 0, 0, 0, 'default', -1, 'default', 'capture_format=YUYV', 'defaul
* else
*     open_framegrabber ('File', -1, -1, -1, -1, -1, -1, 'default', -1, 'default', -1, 'default', 'datacode/ecc200/ecc2
* endif
*
open_framegrabber ('uEye', 1, 1, 0, 0, 0, 0, 'default', 8, 'default', -1, 'false', '/cam/set1', '1', 0, -1, AcqHandle)

```

Figure 5 - uEye cameras with driver software are available for both platforms with identical interfaces.

You can insert the code lines for the uEye camera image acquisition also with the HDevelop assistant "Image Acquisition". Choose "uEye" as interface.

After that, the script could be executed both on the Desktop PC and on the Raspberry Pi. Copy the script on the embedded board and execute it again with the **hrun** tool.

```
pi@raspberrypi:~ $ hrun datacode.hdev
```

As result, you see a live display of your uEye camera. If you hold datamatrix codes of the type ECC200 in front of the camera, HALCON will decode the codes and show you the content.

Your simple Embedded Code Reader is ready to use.

Image acquisition tips

Because the **camera configuration** on the embedded board respectively under HALCON is not really easy, you better use the possibility of camera pre-configuration with a Windows PC and the **uEye Cockpit**. This is the best possible solution to adjust your camera. Save the configuration in the camera's parameter set to use it afterwards in HALCON. This is very simply done when opening the frame grabber, by using **/cam/set1** as parameter value for **"CameraType"**. For further information on uEye camera configuration read our TechTip "Parameterizing instead of Programming: The faster way to camera setup".

Because the embedded board does not have the performance of a Desktop PC, you should reduce the camera framerate for your first test. First try getting a stable camera live display without HALCON before using the camera configuration with your image processing.

Tips for building your embedded development environment

Setup your embedded board starting with preparing the OS image, copying files from place to place until the program execution, could be handled in different ways. With the latest Raspbian image you can work with your Raspberry Pi also completely "headless" (without connecting a display, keyboard and mouse) over SSH and VNC servers. However, there are several obstacles to be taken. Especially in a company network the network connection from your development PC to your Raspberry Pi can be a tremendous challenge.

The easiest way to start is the classical way by connecting a display, a mouse, a keyboard and by using an USB memory stick for copying data (programs, scripts, HALCON installation files) between your two systems. For more complex applications this is not the most comfortable way, but for a first proof of concepts really easy and fast to realize.

More application notes and TechTips can be found at our website https://en.ids-imaging.com/good-to-know.html

Author

Heiko Seitz, Technical writer

Contact

IDS Imaging Development Systems GmbH
Dimbacher Str. 6-8
74182 Obersulm
Germany

Phone: +49 7134 96196-0
Email: marketing@ids-imaging.com
Web: www.ids-imaging.com

Appendix

Source code “datacode.hdev”

Modified code of script **datacode.hdev** to read codes with a uEye camera. The most important modifications are colored. For using the script, copy it in HDevelop und save it as *“datacode.hdev”*

```
dev_update_off()
dev_close_window()

Width  := -1
Height := -1

close_all_framegrabbers ()
open_framegrabber ('uEye', 1, 1, 0, 0, 0, 0, 'default', 8, 'default', -1, \
                   'false', '/cam/set1', '1', 0, -1, AcqHandle)

create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
grab_image_start (AcqHandle, -1)
while (1)
    grab_image_async (Image, AcqHandle, -1)
    if (Width = -1)
        get_image_size (Image, Width, Height)
        open_window (0, 0, Width, Height, 0, 'visible', '', WindowHandle)
    endif
    disp_obj (Image, WindowHandle)
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], ResultHandles,\
                       DecodedDataStrings)
    if (|DecodedDataStrings|)
        set_color (WindowHandle, 'green')
        disp_obj (SymbolXLDs, WindowHandle)
        disp_message (WindowHandle, DecodedDataStrings, 'window', 12, 12, \
                      'black', 'true')
    else
        disp_message (WindowHandle, 'No ECC200 datacode found !', 'window', \
                      12, 12, 'red', 'true')
    endif
endwhile

close_window (WindowHandle)
close_framegrabber (AcqHandle)
```