AN13243

Machine Vision Applications on i.MX RT1050/1060 with OpenMV and Micropython

Rev. 0 — 10 May, 2021 Application Note

Contents

1 Introduction

This application note, using the work on the i.MX RT1060 EVK development board as an example, introduces the porting and adaptation of the open source machine vision project OpenMV. In terms of programming model, OpenMV combines with MicroPython to enable users to use Python language to develop the application of machine vision. You can use Python on this development board to quickly evaluate and use OpenMV functions, or on this basis, customize their own visual processing module and communicate with other modules in the system. For those who are more familiar with the MicroPython and OpenMV software architectures, further customization can be made, such as adding new functionality or stripping out the MicroPython system to develop machine vision applications in a pure C environment. The native project management and build system of OpenMV is based on GCC and make under Linux. To facilitate the development habits of most MCU embedded engineers, the development environment is also migrated to Keil MDK5.

We assume that you have basic experience of development with KEIL MDK, with the knowledge of CMSIS-Pack, the concept of **target** in KEIL and how to switch between them.

NOTE

If you are not yet familiar with how to use Micropython on i.MX RT1050/1060, see *Building Micropython with KEIL and Programming with Python on i.MX RT1050/1060* (document AN13242).

The i.MX RT1050/1060 series processors each has a single Arm® Cortex®-M7 core, which operates at the speed up to 600 MHz. The great processing capability, real-time feature, and reach integration of abundant peripherals make i.MX RT1050 ideal for lot of high-performance applications, such as industrial computing, motor control, power conversion, smart consumer products, high-end audio systems, home and building automation.

2 Hardware platform

2.1 i.MX RT1050/60 crossover process

The i.MX RT1050/60 offered by NXP with single Arm® Cortex®-M7 core can operate at the speed up to 600 MHz. It has 512 KB on-chip RAM, which can be flexibly configured as core Tightly-Coupled Memory (TCM) or general-purpose RAM. RT1060 has additional dedicated 512 KB of OCRAM. It provides:

- Various interfaces for connecting various external memories

- A wide range of serial communication interfaces, such as USB, Ethernet, SDIO, CAN, UART, I2C, and SPI
- Rich audio and video features, including LCD display, basic 2D graphics, camera interface, SPDIF and I2S audio interface
- Other notable features including various modules for security, motor control, analog signal processing, and power management

2.2 i.MX RT1050/60 EVK board

i.MX RT1050 EVKB/1060 EVKB board is a platform designed to showcase the most commonly used features of the i.MX RT1050 processor. The EVK board offers the below features:

- Memory: 256 Mbit SDRAM, 64 Mbit Quad SPI Flash, 512 Mbit Hyper Flash, TF Card Slot
- Communication interfaces: USB 2.0 OTG connector, USB 2.0 host connector, 10/100 Mbit/s Ethernet connector, CAN bus connector
- Multimedia interfaces: CMOS sensor connector, LCD connector
- Audio interfaces: 3.5 mm stereo headphone hack, board-mounted microphone, SPDIF connector (not mounted by default)
- Debug interfaces: On-board debug adapter with DAP-Link, JTAG 20-pin connector
- Arduino interface
- User button and LEDs

The i.MX RT1050 EVKB is as shown in Figure 1.

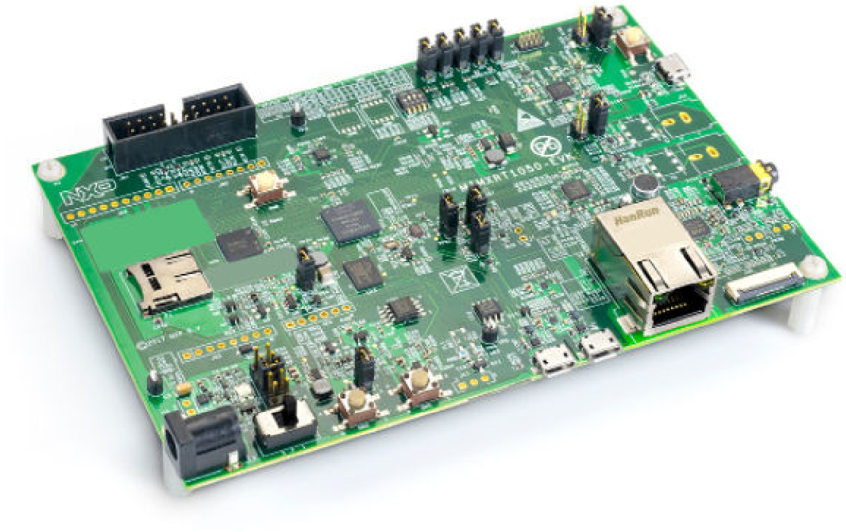


Figure 1. i.MX RT1050 EVKB

3 Machine vision and OpenMV project

3.1 Introduction to machine vision

In a machine vision system,

1. A real image of the object is automatically received and processed through the device and optical non-contact sensor.
2. To obtain the required information or control the robot by analyze the characteristic of the image of color, light and shade, shape characteristics, boundary features, area, length, etc. The measurement and decision-making uses the designed algorithm instead of the human eye.

Traditional machine vision is often used in the working environment with clear boundaries and relatively simple and controllable scene content, such as a corner of the factory production line, fixed monitoring equipment, etc. Recently, with the rise of deep learning, visual processing systems based on deep neural networks are expanding their application fields.

Machine vision technology has been used in industryfor a long time. Due to the large amount of computation, industrial computer has been used as the main computing component. But as the computing power of MCU devices has continued to increase in recent years, some projects to deploy machine vision technology are developed on MCU. OpenMV is one of the excellent MCU side machine vision projects.

3.2 Introduction to OpenMV project

OpenMV stands for **Open Machine Vision**. It is both a company name and a project name. The OpenMV project is about creating a low-cost, scalable, Python-powered programmable machine vision module that can run on an MCU. The goal is to become the **Arduino for machine vision**, bringing machine vision algorithms closer to makers and hobbyists. The OpenMV team has done the difficult and time-consuming algorithm-based work to allow users to focus on their own applications.

3.3 Components of OpenMV project

The OpenMV project consists of following software parts and hardware parts.

- **Hardware - OpenMV Cam**

  This is the hardware carrier of OpenMV functionality, which has been in development for four generations. Its main control chip is an MCU, upgraded from the original 168 MHz Cortex-M4 to a 480 MHz Cortex-M7, which is compatible with both hardware and software. They are very small in shape, just like a larger M12 camera module. After porting and adapting OpenMV to i.MX RT1060 EVK, we also have third-party partners who design and manufacture a visual module similar to OpenMV CAM 3rd generation hardware in appearance.

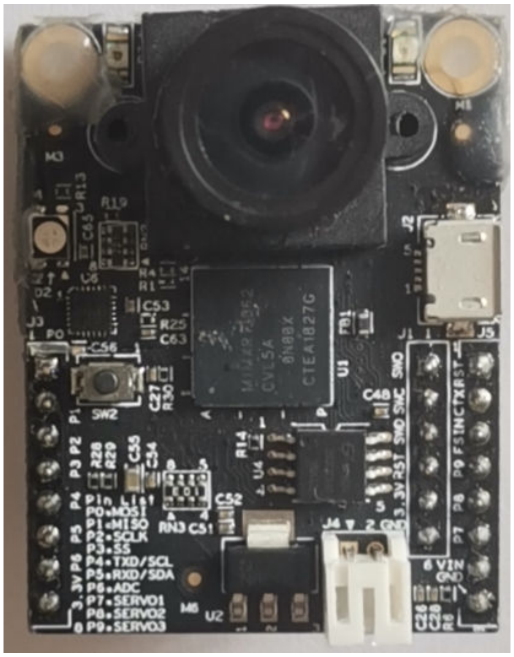


Figure 2. i.MX RT1060 vision module powered by OpenMV MCU side software

- **Software**

  Open source code under the MIT open source license, after build, OpenMV MCU side software will generate a firmware image running on the master MCU of OpenMV CAM. It is the main carrier of software functions of OpenMV project. It implements machine vision algorithm on MCU and can also drive MCU to control peripheral devices. The porting of OpenMV to i.MX RT1050/1060 is also carried out on the MCU side software of OpenMV. Because the OpenMV MCU side software originally ran on a very different MCU platform from i.MX RT1050/1060, this part of the porting costs most effort. The MCU side software of OpenMV is also a customized MicroPython, and its basic functions are provided through the sensor module and the image module.

— OpenMV IDE

This is an integrated development environment specially created by the OpenMV team for developers. It is not only a cross-platform Python code editor, but also can communicate with OpenMV firmware through USB or Wi-Fi through special debugging protocol, preview the current frame buffer in OpenMV firmware, and control the start and stop of Python scripts. The OpenMV IDE also comes with a large number of examples and integrates common machine vision tools.

— OpenMV camera module

This is new from OpenMV H7 cam. In the past, camera chip is soldered on the OpenMV Cam board, but from OpenMV H7 Cam, cameras are designed as pluggable modules, such as OV7725 module, global shutter module, high resolution module, and passive infrared module.

— OpenMV expansion boards

It is connected by two rows of extension ports around OpenMV CAM. OpenMV expansion boards contain a variety of types, such as LCD board, servo board, cloud board, TV board and so on.

4 Build and run OpenMV firmware on your i.MX RT1050/1060 EVK

4.1 Download source code

4.1.1 Download specific version for this AN

Download the source code and unzip it. We strongly recommend to download the version with tag **an_mpy1050_rev1**. This application note describes the operations with this version.

4.1.2 git clone

If you're familiar with Git, you can open a command window, navigate to the directory you want to put in, and then execute the command.

```
git clone https://github.com/RockySong/micropython-rocky.git
```

It will by default clone the **omv_initial_integrate** branch. To get exactly the same result, we strongly recommend to check out to **an_mpy_rt1050_60.**

NOTE

If you meet any issues with latest code, please switch to the **an_mpy1050_rev1** tag.

4.2 Open KEIL project and build firmware

Based on **an_mpy1050_rev1** version, locate the *\ports\prj_keil_rt1060\ mpyrt1060.uvprojx* and open it with KEIL 5.0 or above.

NOTE

This project requires NXP.MIMXRT1062_DFP.12.1.0. It is a CMSIS pack.

This project has multiple targets as shown in Figure 3.

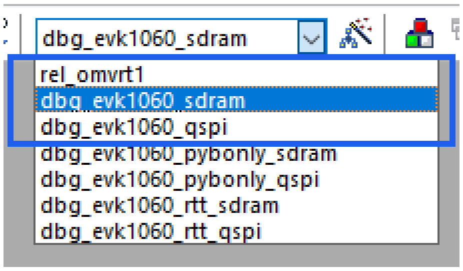


Figure 3. Muitiple targets

The first three targets are designed to generate firmwares with OpenMV functionality.

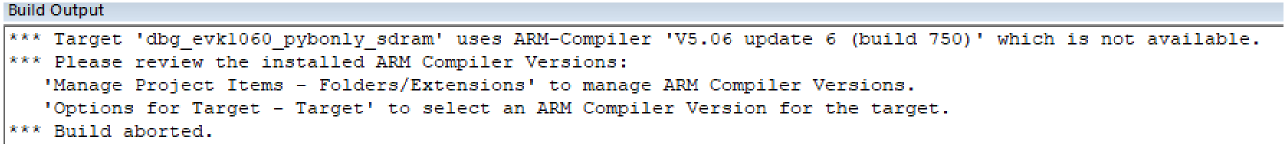
`rel_omvrt1`	Build for the small i.MX RT1060 vision board NOTE Do **NOT** build with this target for i.MX RT1050/1060 EVK boards!
`dbg_evk1060_sdram` (best debugging experience)	Build firmware to run from SDRAM for i.MX RT1060 EVK. Downloading does not need to program flash so it is very fast but re-downloading is required after power cycle or reset. Recommend to re-download for debugging purpose.
`dbg_evk1060_qspi`	Build firmware to run from QSPI Flash for i.MX RT1050/1060 EVK board, good for demo and normal use. Re-download is not required after power cycle or reset, but programming flash may take more time.

The three build targets use i.MX RT1060's on-chip ITCM, DTCM, and OCRAM. It is very important to reasonably put performance critical code and data to them to greatly improve the performance of machine vision algorithms.

Select the **dbg_evk1060_sdram** target，as debugging in SDRAM is most convenient. No matter whichever target you select, click

or press **F7** to build the whole project.

If you get below error:

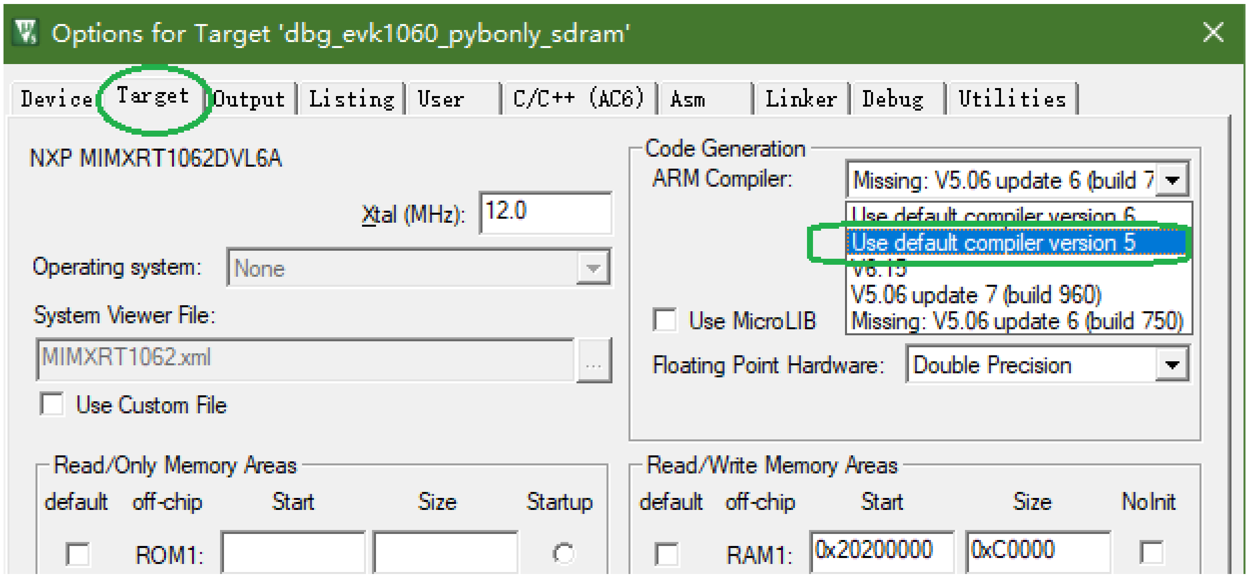


```
Build Output
*** Target 'dbg_evk1060_pybonly_sdram' uses ARM-Compiler 'V5.06 update 6 (build 750)' which is not available.
*** Please review the installed ARM Compiler Versions:
   'Manage Project Items - Folders/Extensions' to manage ARM Compiler Versions.
   'Options for Target - Target' to select an ARM Compiler Version for the target.
*** Build aborted.
```

Figure 4. Error message

It means the specified AC5 compiler version is not found. In this case, click

or press **Alt+F7** to pop out the **Options for Target** dialog. Switch to the **Target** pannel and in the **Code Generation** frame, select the **Use default compiler 5** in the **ARM Compiler** combo box.



NXP MIMXRT1062DVL6A

Xtal (MHz): 12.0

Operating system: None

System Viewer File: MIMXRT1062.xml

□ Use Custom File

Code Generation

ARM Compiler: Missing: V5.06 update 6 (build 7

Use default compiler version 6
Use default compiler version 5
V6.15
V5.06 update 7 (build 960)
Missing: V5.06 update 6 (build 750)

□ Use MicroLIB

Floating Point Hardware: Double Precision

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
	□ ROM1:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
	□ RAM1:	0x20200000	0xC0000	□

Figure 5. Options for Target window

4.3 Prepare for file system

The OpenMV MCU firmware functions are based on the Micropython system, where the file system concept is extremely important and all Python scripts are stored in the file system. In our ported OpenMV system, the file system on the TF/microSD card and the file system in the program Flash are supported. During startup, the system first checks whether the TF card is inserted.

- If so, mount the TF card into the root file system.
- If not, mount a block of approximately 2 MB of space allocated from QSPI Flash as the file system. If it is used for the first time, the system will automatically format this area in the QSPI Flash.

The Flash file system makes it possible to run Micropython without a TF card. However, its write performance is extremely poor (around 10 KB/s) and it does not include wear balance. It is generally recommended to place only files that do not change frequently, such as configuration files, debugged scripts, and so on to internal Flash file system. In addition, when writing data to the Flash file system, the interrupt will be turned off, affecting real-time. Therefore, it is highly recommended to insert a TF card formatted using FAT/FAT32 on the board.

4.4 Download and run

To download, perform the following steps:

1. Determine which debugger you are using. i.MX RT1060 EVK has an on-board CMSIS-DAP compatible debugger, but due to the large firmware generated by this project, the download with CMSIS-DAP is slow. J-Link can also be used. On the i.MX RT1060 EVK, to use J-Link, disconnect J47 and J48; or shortcut them if you use the onboard CMSIS-DAP compatible debugger. Under Keil, J-Link downloads far faster than the on-board CMSIS-DAP. As the firmware has grown several times since the inclusion of OpenMV, so it can save lots of time to download it using J-Link.

   NOTE

   If J-Link is used, after downloading to SDRAM, click the **Reset** button again before executing the program. Otherwise, a hard fault may accidentally occur.

2. Connect the i.MX RT1060 EVK, as shown in Figure 6.

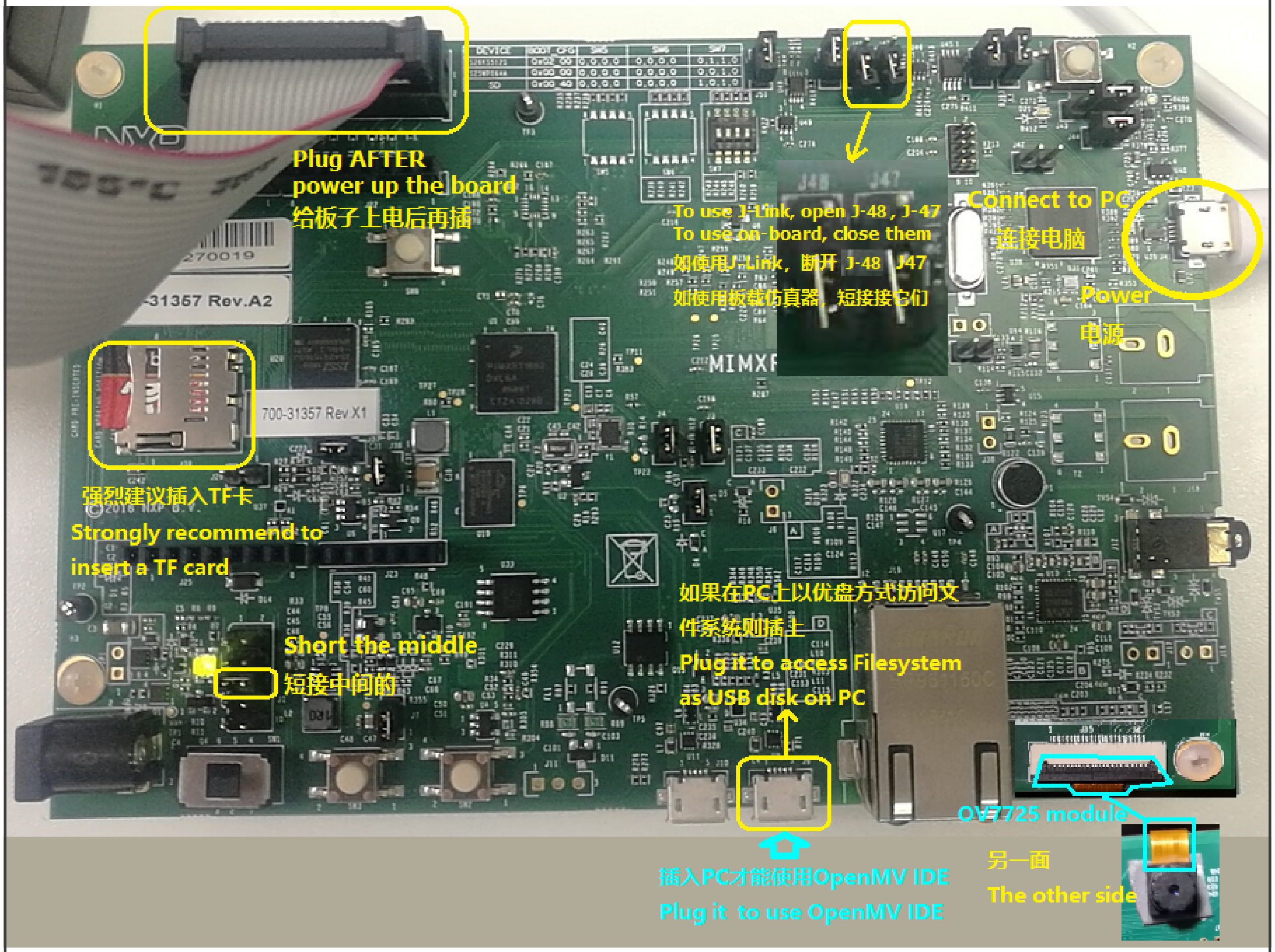

Plug AFTER power up the board
给板子上电后再插
To use J-Link, open J-48 , J-47
To use on-board, close them
如使用J-Link，断开 J-48 J47
如使用板载仿真器，短接接它们
Connect to PC
连接电脑
Power
电源
强烈建议插入TF卡
Strongly recommend to insert a TF card
Short the middle
短接中间的
如果在PC上以优盘方式访问文件系统则插上
Plug it to access Filesystem as USB disk on PC
OV7725 module
另一面
The other side
插入PC才能使用OpenMV IDE
Plug it to use OpenMV IDE

Figure 6. Connecting i.MX 1060 EVK

NOTE

The lower-right corner is connected to an OV7725 module, which is the input source of the video signal. If you also have the LCD module for i.MX RT1060 EVK, you can connect it, so that you can preview the image on the LCD module and overlay the annotation after OpenMV processing.

The USB port is near the Ethernet. The OpenMV IDE communicates with the development board through this interface.

3. Open a serial port terminal and connect to the virtual serial port presented by the onboard debugger on the development board with the baud rate set to **115200**.

To run the i.MX1060 EVK, perform the following steps:

- To run from SDRAM:
  1. Click or press **Ctrl-F5** after build.
  2. Wait until program code is fully downloaded and KEIL shows debug toolbar.
  3. Press **F5** to run.
- To run from QSPI Flash (XIP):

1. Click LOAD or press **F8** to download the built firmware to Flash.

- To run from Flash:
  1. Directly reset the board to run. There is no need to enter the KEIL’s debug session.

5 Access Micropython file system from PC

To access the contents of the file system on your PC, connect to the computer on a USB port near the Ethernet port, which will present a removable disk on the computer.

- When no TF card is inserted, the contents of the Flash file system are displayed on the removable disk.
- When one TF card is inserted, the contents of the TF card are displayed on the removable disk.

When accessing the Flash file system, the write speed is only about 10 KB/s. When accessing the TF card, the general read and write speed is about 10 MB/s. OpenMV can record MJPEG video and store images to the file system. When using these functions, be sure to insert a TF card.

NOTE

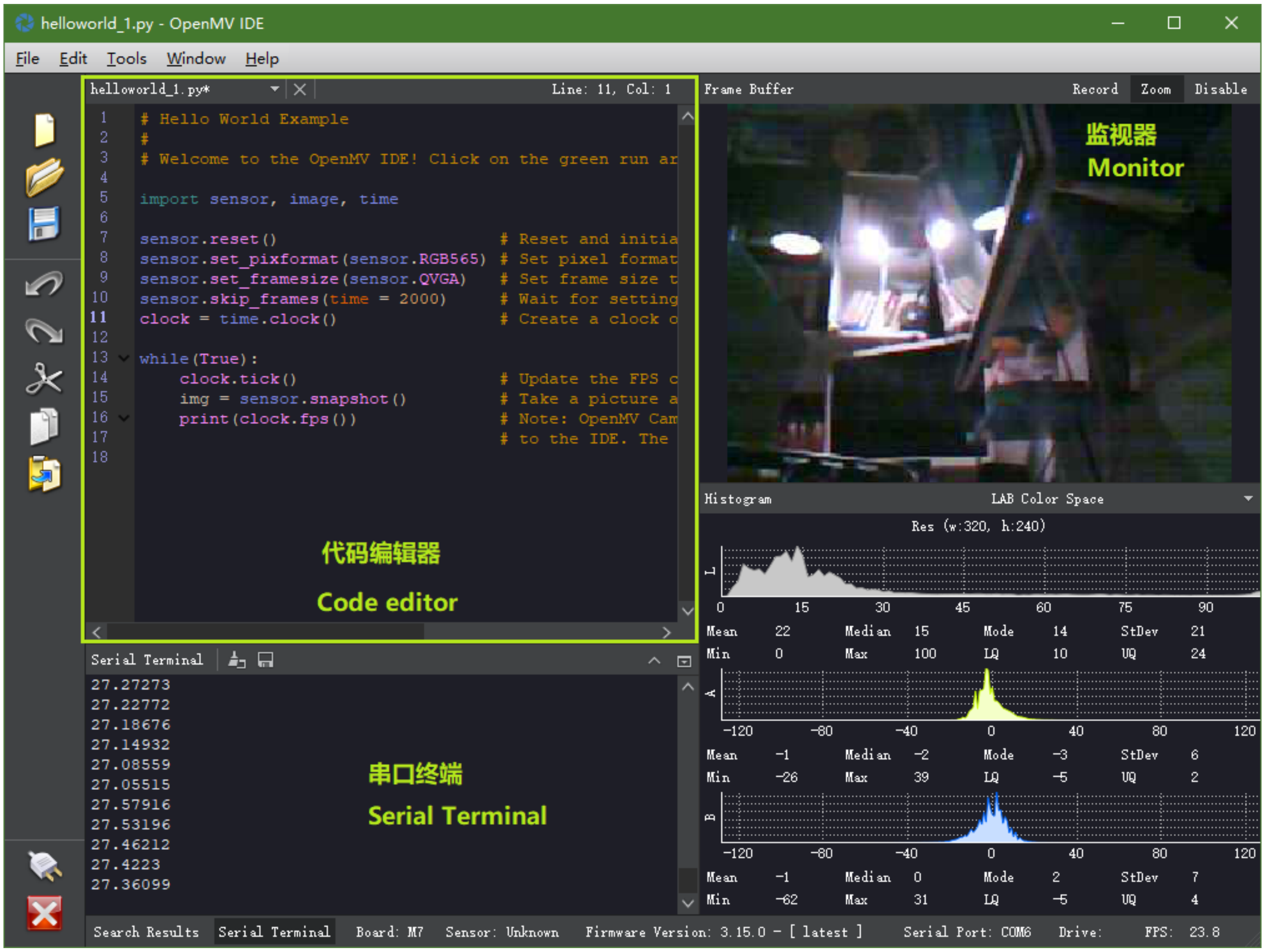
Do NOT include a directory named **flash** (case sensitive) in your TF card. This will cause your Python script to access `/flash` while still accessing the Flash file system and seeing the contents of the TF card on your PC.

6 Explore OpenMV functions with OpenMV IDE

OpenMV treats Micropython as its runner, so it is natural to use MicroPython development tools to develop OpenMV applications. However, there are some special requirements for machine vision applications, such as real-time preview of the collected and annotated images, analysis of the basic statistical characteristics of the image, real-time control of the script to start and stop, integration of common machine vision tools, etc. Both low-bitrate control flow and high-bitrate image data flow are involved.

To meet this new demand, the OpenMV project, besides providing the core machine vision function, has developed a new set of development and debugging mechanism. It enables the PC to monitor and control the running state of OpenMV firmware through the USB virtual serial port. OpenMV team also developed a supporting IDE environment, called OpenMV IDE. When porting the OpenMV MCU side software to i.MX RT1060 platform, we also ported this capability to communicate with OpenMV IDE, so we can still use OpenMV IDE to monitor the running state of OpenMV MCU side software on i.MX RT1060.

OpenMV IDE can be downloaded from https://openmv.io/pages/download. It supports Windows, OSX, Ubuntu, and Raspberry Pi. OpenMV IDE contains the code editor window, framebuffer preview window, serial terminal window, and image histogram window, as shown in Figure 7.


helloworld_1.py - OpenMV IDE
File Edit Tools Window Help
helloworld_1.py*
Line: 11, Col: 1
Frame Buffer
Record Zoom Disable
监视器
Monitor
Histogram
LAB Color Space
代码编辑器
Code editor
Serial Terminal
串口终端
Serial Terminal
Search Results Serial Terminal Board: M7 Sensor: Unknown Firmware Version: 3.15.0 - [latest] Serial Port: COM6 Drive: FPS: 23.8

```
# Hello World Example
#
# Welcome to the OpenMV IDE! Click on the green run ar

import sensor, image, time

sensor.reset()                      # Reset and initia
sensor.set_pixformat(sensor.RGB565) # Set pixel format
sensor.set_framesize(sensor.QVGA)   # Set frame size t
sensor.skip_frames(time = 2000)     # Wait for setting
clock = time.clock()                # Create a clock o

while(True):
    clock.tick()                    # Update the FPS c
    img = sensor.snapshot()         # Take a picture a
    print(clock.fps())              # Note: OpenMV Cam
                                    # to the IDE. The
```

```
27.27273
27.22772
27.18676
27.14932
27.08559
27.05515
27.57916
27.53196
27.46212
27.4223
27.36099
```

```
Res (w:320, h:240)
Mean  22   Median  15   Mode  14   StDev  21
Min   0    Max     100  LQ    10   UQ     24
Mean  -1   Median  -2   Mode  -3   StDev  6
Min   -26  Max     39   LQ    -5   UQ     2
Mean  -1   Median  0    Mode  2    StDev  7
Min   -62  Max     31   LQ    -5   UQ     4
```

Figure 7. OpenMV IDE

NOTE

The refresh rate of the real-time refresh frame buffer monitor window on the right side of the lower status bar is often lower than the cycle rate of the main image processing loop. In addition, since image compression and uploading also occupy a part of MCU terminal system resources, if the image monitor window is disabled, the image processing frequency will also increase.

The FPS displayed at the right bottom corner is the image refresh rate of OpenMV IDE, typically less than the real image refresh rate inside device. The image compression and upload take some MCU side resources, so if you disable the frame buffer window, image processing rate will grow.

For more details about OpenMV IDE, see OpenMV IDE Overview.

To communicate with the OpenMV IDE, there is also a dedicated module called USBDBG on the OpenMV MCU terminal software.

In our ported OpenMV MCU terminal software, USBDBG and REPL multiplexed the same USB virtual serial port, and the USB interface (J9) near the Ethernet port on i.MX RT1060 EVK was needed to connect to the computer through USB cable before use. **REPL via USB cannot be used during the connection to the OpenMV IDE is active**.

7 Skeleton of OpenMV machine vision applications

To run the machine vision algorithm, perform the following steps:

1. Prepare the camera and then loop through the image data.

Let's take a small program that implements this **dry run** function as an example, which can also be considered as the **Hello World** program in OpenMV machine vision.

a. Import the required modules.

```
import sensor, image, time
```

sensor and **image** are the two most important modules in OpenMV. The sensor module encapsulates the camera operation and the image module encapsulates the operation of machine vision. The time module can be used to measure the frame rate.

b. Prepare the camera.

```
sensor.reset() # Reset and initialize the sensor.
```

The `sensor.reset()` method resets and initializes the controller chip of camera, such as OV7725. Different sensors take different time to reset.

c. After the camera is initialized, give the desired resolution, color/grayscale mode Settings. For this simple example, use QVGA@RGB565, which is also the setting used for most examples in OpenMV.

```
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565
sensor.set_framesize(sensor.QVGA)   # Set frame size to QVGA (320x240)
```

d. Use the **set_pixFormat ()** method above to set the pixel format. The color format commonly used is RGB565.If you want a grayscale, change the **RGB565** to **GRAYSCALE**.

e. After working, the sensor chip often needs 1-2 seconds to dynamically adapt to the ambient lighting. Adjust the exposure time, white balance and other parameters. In OpenMV, you can optionally skip images during this transition.

```
sensor.skip_frames(time = 2000) # Wait for settings take effect.
```

f. Use the **skip_frames()** method above ton skip images during a specified time, in milliseconds. In the above code, 2000 ms is skipped.

Now, the camera has been working normally.

2. Enter the main working loop.

Assuming that the system is working all the time, we use an infinite loop. Once in a cycle, take a picture and do the actual processing on that picture.

```
while(True):
            clock.tick()                      # Update the FPS clock.
            img = sensor.snapshot()           # Take a picture and return the image.
            <do further image processing>
            print(clock.fps())                # Note: OpenMV Cam runs slower when connected to the IDE. The FPS should increase once
disconnected.
```

The `sensor.snapshop()` above takes one snapshot and returns the image.

The above is how a **Hello World** of the machine vision application is realized through OpenMV, which can be regarded as the starting point for adding subsequent machine vision logic.

You can also see from the main loop above that images are processed individually in OpenMV.

8 Image capture configurations

Image acquisition is the eye of machine vision system. High-quality acquisition effect provides a strong guarantee for the correctness of machine vision algorithm results. Developers need to reasonably set the image resolution, color mode, color and brightness control parameters according to the needs of the machine vision task.

8.1 Color mode

For image capture, OpenMV supports the **.RGB565** sensor as color and **.GRAYSCALE** as grayscale, used in the **sensor.set_pixformat()** method.

When binary image is needed, use the **binary()** method of image object to convert to binary image.

8.2 Image resolution and window clipping

Although the extremely high resolution of 1920 × 1080 or even 3840 × 2160 (4 K) on computers was used for nowadays' desktops, 320 × 240 (QVGA) or even lower is the most commonly used in OpenMV machine vision applications, as show in Table 1.

Table 1. Image resolution

Resolution	Abbreviation/Remark
128 × 128	B128 × 128
160 × 120	QQVGA
320 × 240	QVGA
352 × 288	CIF
480 × 272	i.MX RT1060 EVK LCD resolution
640 × 480	VGA

With the `sensor.set_windowing()` method, you can clip a smaller window from a larger resolution input. For example, 480 × 272 is not a native resolution supported by OpenMV, but can be clipped from the VGA resolution.

In the ported camera driver, the window clipping is done at the same time as the image is received, and the original image does not take up an additional frame buffer. The `sensor.set_windowing()` method is for clipping. It can accept both the (left, top, length, and width) of the target rectangle position & size given by the 4-tuple, and the **cocentric** rectangular window from the clipped image given by only the (length, width) size of the target rectangle. For example, we can crop out a window with an i.MX RT1060 EVK LCD screen from the center of the original image as follows:

```
sensor.set_framesize(sensor.VGA)
sensor.set_windowing((480,272))
```

NOTE

The parentheses around (480,272) are required to indicate that the argument is passed as a tuple. The `sensor.set_windowing()` method determines whether there are only two or four elements in the tuple to determine whether the rectangular position is manually specified.

8.3 Brightness control

The brightness of images has a significant impact on the quality of machine vision algorithms. The combination of camera exposure time and ambient lighting greatly affects the brightness of images. Photographic chips generally provides a variety of programmable direct or indirect control that affects brightness in the captured image. In OpenMV, Python API bindings provide a variety of ways to directly and indirectly control brightness, briefly described as below:

- Adjust brightness directly

  With the `sensor.set_brightness()` method, you can directly control the brightness logic of the sensor. Configurable range is an integer between **[-3,3]**.

- Increase exposure time length

  With the `sensor.set_auto_exposure` (False, integer) method, you can manually specify an exposure time, ranging between [0, 10000]. The bigger value indicates longer exposure time and higher brightness.

- Increase the gain ceiling

  With the `auto-gain mode` method (by default after sensor reset), increase the gain ceiling up to 128 by *sensor.set_gainceiling(2/4/8/16/32/64/128)*. The higher the gain, the brighter the image.

- Specify gain manually

  With the `sensor.set_auto_gain` (**False**, 2/4/8/16/32/64/128) method, to manually specify the desired gain.

- Lower the master clock (MCLK) of the sensor

  With the `sensor.set_framerate()` method, adjust the main clock frequency of the photosensitive chip and lower the frequency to increase the exposure time and brightness. In our ported system, the clock source and frequency divider can be selected by the user, encoded as follows:

  ```
  {encoded clock source}<<9 | {clock_divider+1}<<11
  ```

  The encoded clock sources can be:

  - 0 : 24 MHz
  - 2 : 120 MHz

  `clock_divider` can be 0 to 7, which means to divide by 1 to 8.

  For the OV7725 image sensor connected to the i.MX RT1060 EVK, the max allowed input clock is **15 MHz**, multiplied by 4 internally. To decrease OV7725 master clock and increase brightness indirectly, change the clock source to 0 (24 MHz) and set **clock_divider** greater than 1.

- Use VGA as native resolution and clip your window

  OV7725 gets brighter image when working under VGA mode, but high resolution requires more resources. To solve this problem by window clipping, use the following settings to emulate the effect of `sensor.set_framesize` (sensor.QVGA):

  ```
  sensor.set_framesize(sensor.VGA)
  sensor.set_windowing((320,240))
  ```

NOTE

The region of image is only the 50% of original visual range.

The effects of the above methods can be superimposed to make the image brightness vary greatly. Figure 8 shows the image effect in the same dark room, with the lighting source as 3 W white LED strip lamp.

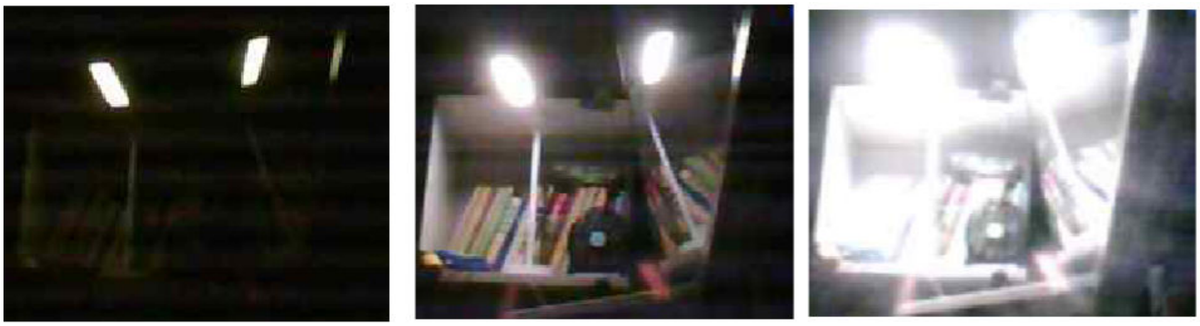


Figure 8. Brightness control effect

As shown in Figure 8, the brightness will affect the color of the image.

9 Image module - Brain of machine vision

OpenMV MCU side software includes the implementation of a rich set of machine vision algorithms. These algorithms are encapsulated in the **image** Python module, responsible for presenting various machine vision functions of OpenMV. If the sensor module introduced above is the **eye** of OpenMV, the image module is the **brain** of OpenMV. To demonstrate what this brain can do, the OpenMV IDE comes with a number of sample programs. The below will describe the commonly-used machine vision functions in the image module with examples.

9.1 Drawing

Draw arrows, lines, crosses, circles, ellipses, key points, and output strings. These functions are not part of machine vision, but can be used to annotate the results of processing and make a simple man-machine interface.

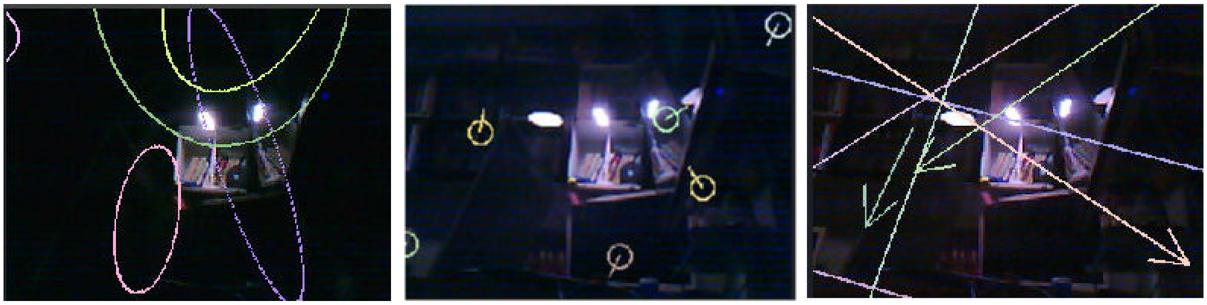


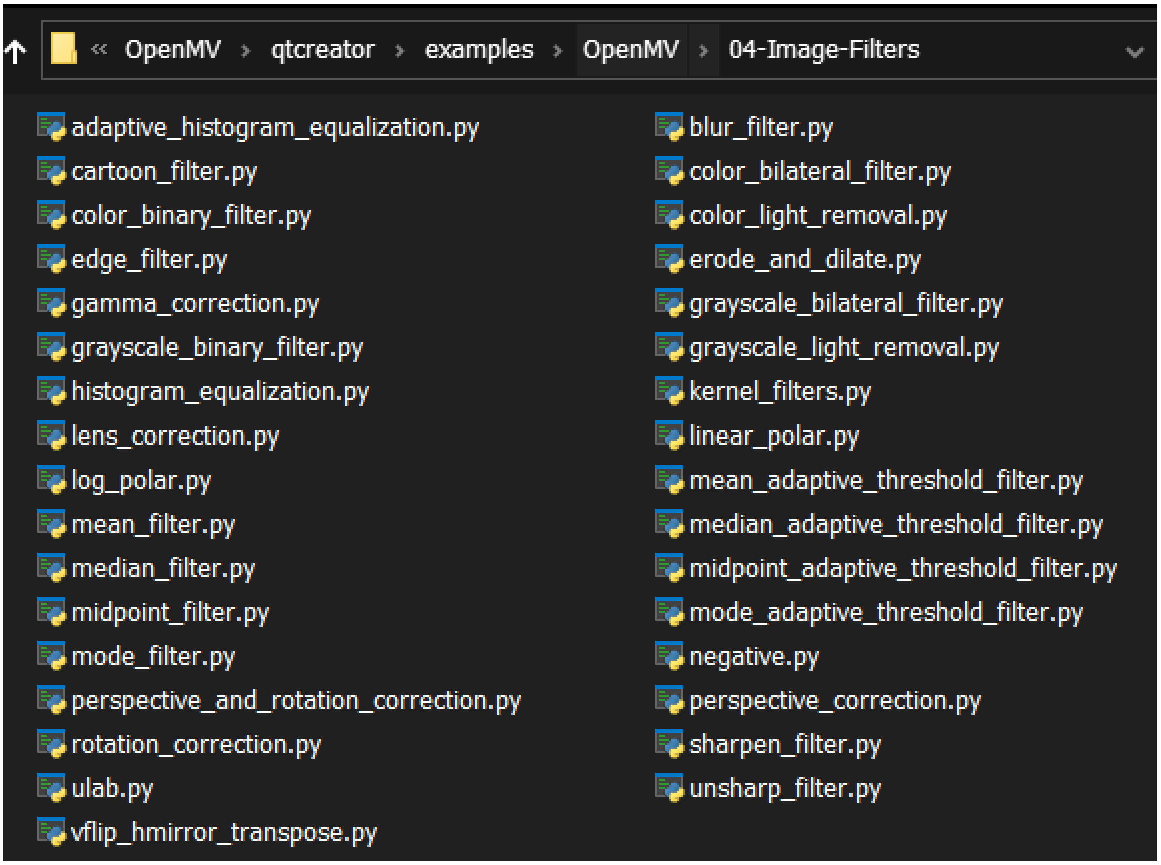
Figure 9. Drawing

9.2 Image processing/filtering

This part contains many common image processing algorithms, including: adaptive histogram, blur, cartoon, bilateral, binary filter, light removal, edge detection, corrosion and expansion, gamma correction, camera calibration, linear kernel function and morphology filter - polar coordinates, the log - polar coordinates, average adaptive threshold filtering, adaptive threshold average filtering and median filtering, median filtering, midpoint adaptive threshold filtering, midpoint filtering, negating, perspective and rotation correction, sharpen, flip and mirror, etc.

Examples related to image processing/filtering are located at *.\OpenMV IDE\share\ QtCreator\Examples \OpenMV\ 04-image-filters*.

You can explore them with Windows File explorer as below:



adaptive_histogram_equalization.py
cartoon_filter.py
color_binary_filter.py
edge_filter.py
gamma_correction.py
grayscale_binary_filter.py
histogram_equalization.py
lens_correction.py
log_polar.py
mean_filter.py
median_filter.py
midpoint_filter.py
mode_filter.py
perspective_and_rotation_correction.py
rotation_correction.py
ulab.py
vflip_hmirror_transpose.py
blur_filter.py
color_bilateral_filter.py
color_light_removal.py
erode_and_dilate.py
grayscale_bilateral_filter.py
grayscale_light_removal.py
kernel_filters.py
linear_polar.py
mean_adaptive_threshold_filter.py
median_adaptive_threshold_filter.py
midpoint_adaptive_threshold_filter.py
mode_adaptive_threshold_filter.py
negative.py
perspective_correction.py
sharpen_filter.py
unsharp_filter.py

Or, open each per time within OpenMV IDE by clicking **File** -> **Examples** -> **OpenMV** -> **Image** -> **Filters**.

9.3 Snapshot

OpenMV supports snapshot on condition or time elapse. Examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\05-Snapshot*.

9.4 Video recording

OpenMV can record small videos in GIF and MJPEG formats. The accompanying example demonstrates that the video recording is triggered by a face detection event and by a movement detection event. Examples are located at *.\openMV IDE\ Share\QTCreator\Examples\openMV\06-video-recording*.

9.5 Face and eye detection

OpenMV includes face and eye detection models trained by the classical machine learning Haar-Cascade method. Relevant examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\07-Face-Detection* and *.\OpenMV IDE\share\qtcreator\examples\OpenMV\08-Eye-Tracking*.

9.6 Feature detection

OpenMV includes a variety of image feature detection functions, including edge detection, circle detection, line segment detection, rectangle detection, directional gradient histogram (HOG), key points, linear regression, and Local Binary Partten (LBP).

Relevant examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\09-Feature-Detection*.

9.7 Color tracking

OpenMV includes a variety of color blob and line tracking functions, including grayscale and color blob tracking, and an example of a vehicle tracing along a black lead line. It tracks a variety of colors at the same time.

Relevant examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\10-Color-Tracking*.

9.8 Barcode and QR code detection

OpenMV has ported the `zbar` library, which can use Python interface to detect single or multiple barcodes and QR codes in the image field of view, and correct lens distortion before detection.

Relevant examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\16-Codes*.

9.9 April-Tag detection

OpenMV detects April-Tag of multiple families and decodes the code number, distance information, attitude information. Multiple April-Tags from different families can be detected in the same field of view.

OpenMV IDE also comes with a small tool to generate April-Tags.

OpenMV supports below April-Tag families: 16H5, 25H7, 25H9, 36H10, 36H11.

Relevant examples are located at *.\OpenMV IDE\share\qtcreator\examples\OpenMV\26-April-Tags*.

10 Get more information

OpenMV has a very complete document system, MicroPython documentation, where you can find documentation about OpenMV and an introduction to the MicroPython language, libraries, and somefeatures of the development board for reference.

Chinese users can refer to Singtown for the OpenMV documents that Sing-town has translated and numerous instructional videos.

11 References

Following documents may offer further reference.

- MicroPython documentation
- MicroPython
- micropython-rocky
- OpenMV
- Singtown

12 Revision history

Revision number	Date	Substantive changes
0	10 May, 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10 May, 2021
Document identifier: AN13243