White paper about the block scan mode with mvBlueFOX3

Verification of mvBlockScan in mvBlueFOX3

1 Introduction

Unlike area scan mode, block scan mode enables inspection of round/rotating body or long/endless materials at high speed (like line scan cameras). The block scan mode acquires an Area of Interest (AOI) block which consists of several lines. The user defines the amount of AOI blocks which are used to create one image. This minimizes the overhead, which you would have instead when transferring AOI blocks as single images using the USB3 Vision and GigE Vision protocols.

This document shows, how you can work with the block scan mode together with mvBlueFOX3 cameras with IMX global shutter sensors from Sony. Triggered by an incremental encoder, the camera in block scan mode displays objects on a rotating drum.

NOTE
Of course, other hardware (next to a conveyor belt, for example) or software trigger sources work as well. In this case you would not need the "**EncoderControl**" and you would have to adapt the "**TriggerSource**". However, in this document we use the incremental encoder as the trigger source.

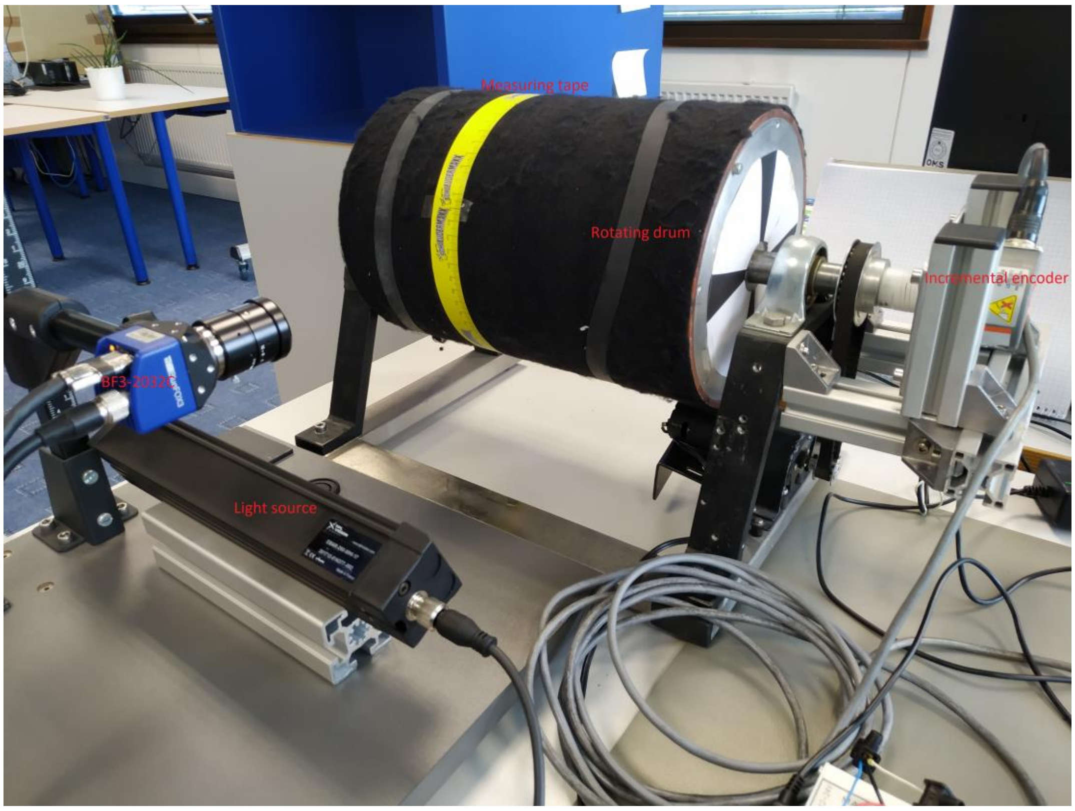
Following topics are being addressed in this document:

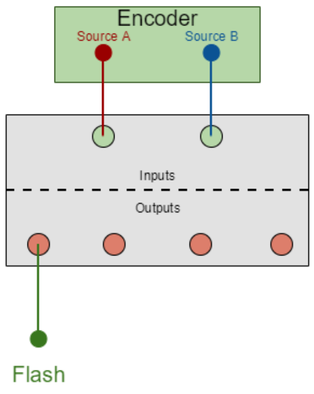
1. How to configure block scan mode with incremental encoder in wxPropView?
2. How to capture a seamless image of a rotating object?
3. What if the object suddenly stops rotating and there are still some blocks in the image buffer?
4. What if the object suddenly starts rotating towards the opposite direction and then back to its original direction?
5. What is the line rate for
    a. mvBlueFOX3-2032C/G,
    b. mvBlueFOX3-2051C/G, and
    c. mvBlueFOX3-2089C/G
    respectively?
6. What are the advantages of block scan mode over line scan camera?

MATRIX VISION GmbH | Talstrasse 16 | DE - 71570 Oppenweiler
Phone: +49-7191-9432-0 | Fax: +49-7191-9432-288 | www.matrix-vision.de
Mail: info@matrix-vision.de

Subject to change without notice / Technische Änderungen vorbehalten – 10/2019

2 Mechanical setup





Materials:

- 1 x rotating drum with a motor
- 1 x measuring tape
- 1 x incremental encoder (1000ppr)
- 1 x camera (mvBlueFOX3-2032C)
- 1 x light source; 1 x trigger box

The rotating drum is mounted with a motor and an incremental encoder. Its surface is covered by a measuring tape for reading out scales *(see the image above on the left)*. The encoder and the light source are connected to the camera's I/O through a trigger box *(see the image above on the right)*. (Encoder) Source A → (Camera) Line 4; (Encoder) Source B → (Camera) Line 5.

3 Procedures

3.1 How to configure block scan mode with incremental encoder in wxPropView?

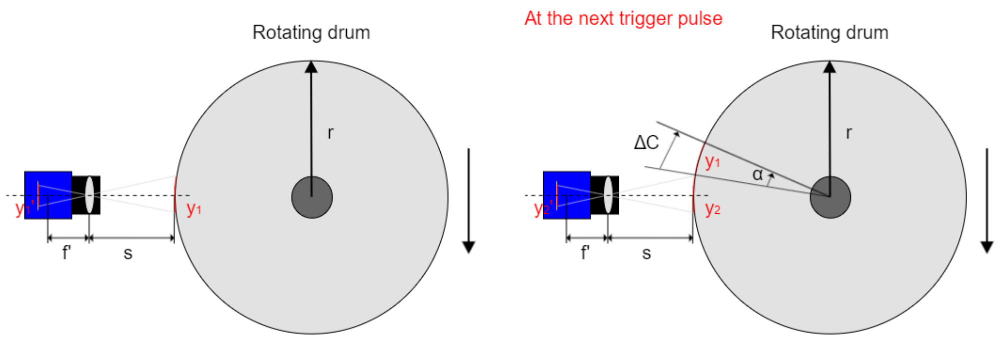
Firmware requirement: version **2.35** or newer.

Configuration steps in wxPropView:

1. Setting → Base → Camera → GenICam → DeviceControl → **DeviceScanType**: mvBlockScan.
2. Setting → Base → Camera → GenICam → ImageFormatControl: configure **OffsetY**, **mvBlockscanLinesPerBlock**(>= 16), **mvBlockscanBlockCount**(>= 2).
3. Setting → Base → Camera → GenICam → ImageFormatControl → **PixelFormat**: BayerRG8 *(for color camera)*.
4. Setting → Base → Camera → GenICam → EncoderControl → **EncoderSourceA**: Line4; **EncoderSourceB**: Line5; **EncoderDivider**: 1; **EncoderOutputMode**: PositionDown/DirectionDown.
5. Setting → Base → Camera → GenICam → AcquisitionControl → **TriggerSelector**: FrameStart; **TriggerMode**: On; **TriggerSource**: Encoder0.
6. Setting → Base → Camera → GenICam → AcquisitionControl → **ExposureTime**.
7. Setting → Base → Camera → GenICam → DigitalIOControl → **LineSelector**: the output line physically connected with the light source; **LineSource**: ExposureActive.
8. (optional) Setting → Base → Camera → GenICam → DigitalIOControl → set **debounce time** for Line4 and Line5.

3.2 How to capture a seamless image of a rotating object?

At every certain turning angle of the rotating drum, a pulse is generated by the incremental encoder. Each pulse (when encoder divider = 1) triggers the camera to capture a block of image, which is then placed one after the other to form a whole image of the rotating drum. The following sketch shows the cross section of the inspection system:



When a new trigger pulse is generated, the rotating drum has turned angle α which covers ΔC of the circumference. Since r is the radius of the rotating drum, n is the encoder divider which defines at how many incremental pulses will a trigger pulse be generated, p is the encoder's pulse rate (*ppr: pulse per revolution*), it is obvious that ΔC is:

$\Delta C = 2*\pi*r*n/p$ *(eq-1)*

In order to have a seamless image consisting of different blocks (y_1', y_2'......), the height of the field of view for each block has to be identical to ΔC:

$y1 = y2 = \Delta C$ *(eq-2)*

Since in this system we used a measuring tape around the rotating drum, it is easy to read out the object height y*(mm)* and the corresponding image height *y*'*(pixel)*. Thus the magnification of the lens β *(pixel/mm)* can also be easily read out:

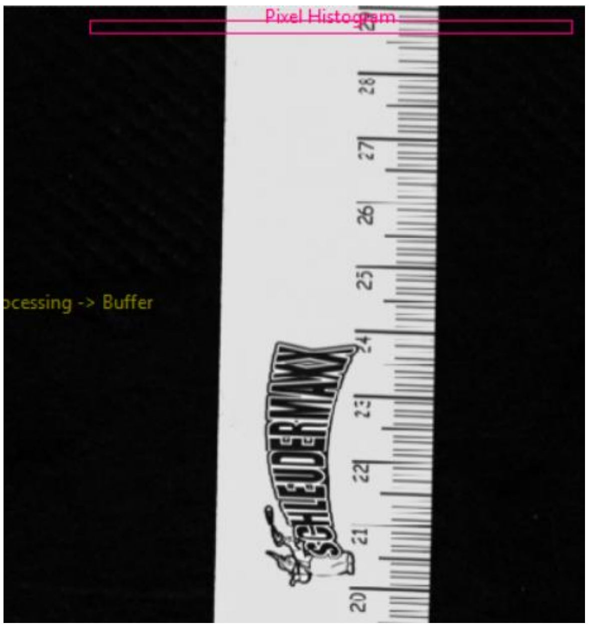
$\beta = y'/y$ *(eq-3)*

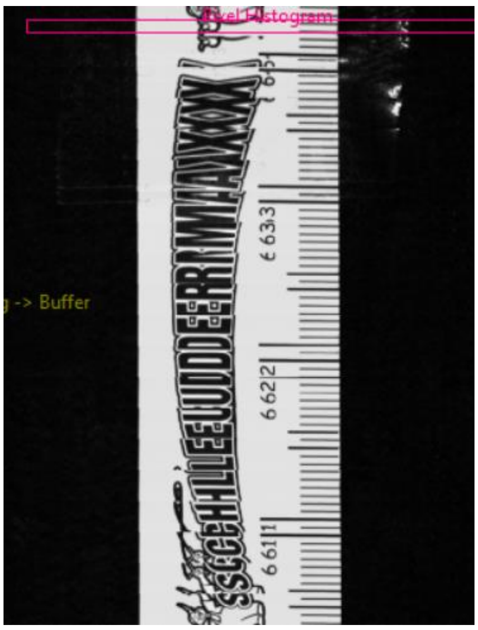
After combining *eq-1, eq-2* and *eq-3*, the required ***lines per block*** (y_1' or y_2') for a seamless image in this system setup should be:

LinesPerBlock $= (2*\pi*r*\beta/p)*n$ *(eq-4)*

The calculation has been validated. For example, image generated from actual *EncoderDivider in wxPropView =**3*** and *LinesPerBlock = **20*** is **seamless**. See the test result below:

If LinesPerBlock is too small, the generated image has **gaps**/pattern loss among blocks *(see the image below on the left)*. If LinesPerBlock is too big, the generated image has **overlaps** among blocks *(see the image below on the right)*.





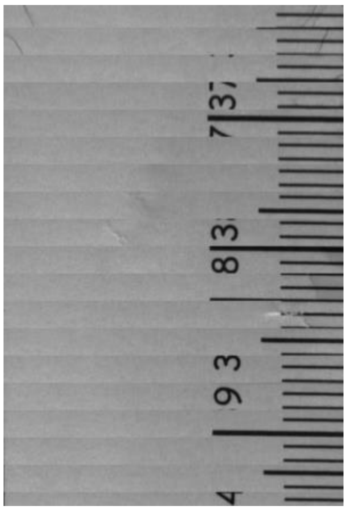
If in other applications the minimum lines per block (e.g. **16**) is required, then *β* and *n* will have to be adjusted to form a seamless image:

β = (LinesPerBlock*p/(2*π*r))*(1/n) *(eq-5) (derived from eq-4)*

Because *β* can be roughly calculated as ***β = |f'/(f'-S)|*** using Gaussian thin lens formula, *β* can be varied by adjusting the focal length *f'* and the working distance *S*.

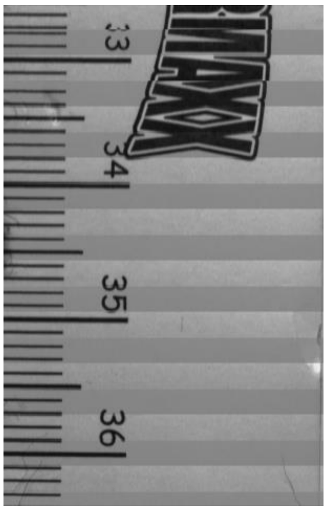
3.3 Direction of rotation

In this example the object moves upwards, as seen from the perspective of the camera. If we placed the camera on the opposite side of the rotating drum, the object would move downwards without actually changing the direction of rotation. The incremental encoder would still count downwards. We'd get an image like this:



The blocks are apparently joined in the wrong order. This can be compensated for by mirroring each block.

1. Setting → Base → Camera → GenICam → ImageFormatControl → ReverseX
2. Setting → Base → Camera → GenICam → ImageFormatControl → ReverseY



The resulting image is inverted and thus not true to life. This might be problematic in some applications. Therefore we recommend to mount the camera in such a way that the object moves upwards in relation to the sensor, as described above.

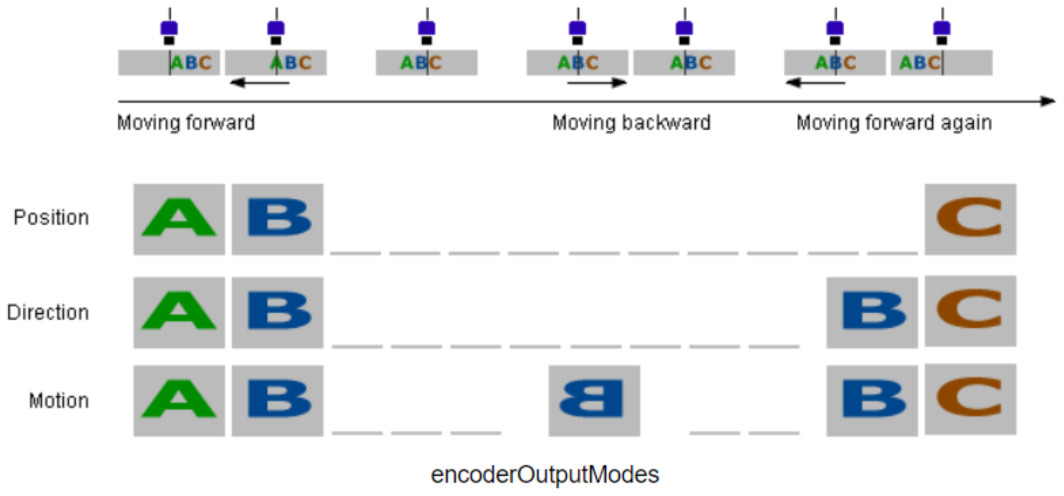
3.4 What if the object suddenly stops rotating and there are still some blocks in the image buffer?

While acquisition the sensor keeps sending blocks to FPGA which collects the blocks to form a frame/image till it reaches the given BlockCount. The complete frame/image is then sent to the image buffer in the driver. If an object suddenly stops rotating while acquisition and there are still several blocks stored in the FPGA. If we don't delete these blocks, once starting inspecting another object, the blocks of the new object will add to the previous blocks till the frame is complete. Thus we will obtain an image of two different objects which is unwanted. Therefore using **Abort** in wxPropView or **imageRequestReset** from our SDK while acquisition to ask the driver to quit receiving incomplete frame from the FPGA would be ideal before the inspection of another object starts.

The **Abort** command takes around 200us to be executed if the host is not heavily loaded.

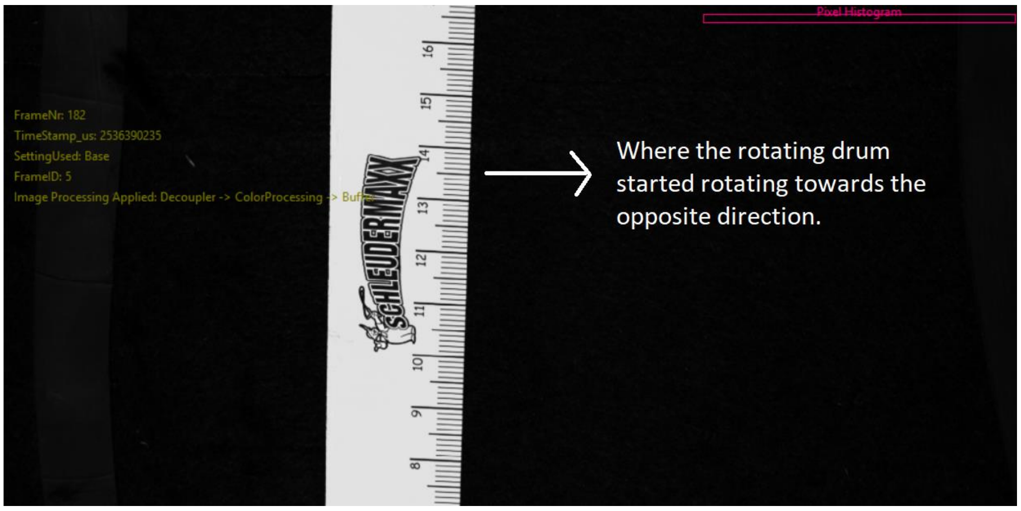
3.5 What if the object suddenly starts rotating towards the opposite direction and then back to its original direction?

In our firmware there are 3 kinds of encoder output modes: position up/down, direction up/down, motion. These 3 modes are well explained here: https://www.matrix-vision.com/manuals/SDK_CPP/classmvIMPACT_1_1acquire_1_1GenICam_1_1EncoderControl.html#a129fb01bf3edfcd0fe6b66197e4284de. A graphical explanation is shown below:



(From C++ SDK)

If we choose position up/down, the image stops when the object starts rotating towards the opposite direction. When it rotates back to its original direction, the blocks starts adding up again from where it stopped. So the acquisition won't be affected by the wrong rotating direction. It has been validated in our test:


Pixel Histogram
Where the rotating drum started rotating towards the opposite direction.

```
FrameNr: 182
TimeStamp_us: 2536390235
SettingUsed: Base
FrameID: 5
Image Processing Applied: Decoupler -> ColorProcessing -> Buffer
```

3.6 Line rate for mvBlueFOX3-2032C/G, mvBlueFOX3-2051C/G, and mvBlueFOX3-2089C/G respectively

mvBlueFOX3-	2032C/G	2051C/G	2089G/C
Line rate (kHz) @ 16 lines/block (kHz)	56.0	48.5	23.6
Block rate (kHz) @ 16 lines/block (kHz)[3]	3.5	3.1	1.5
Max. exposure time (us) @ 16 lines/block (kHz)[1][2]	199	228	440

*[1] **max. exposure time (us)** @ 16 lines/block (kHz) can be observed when **mvAcquisitionFrameRateLimitMode** is set to **mvDeviceMaxSensorThroughput**.*

*[2] **max. exposure time (us)** @ 16 lines/block (kHz) can also be calculated by **1/BlockRate - t_p**, where **t_p** is **Next Trigger Fall Prohibition Time** between two neighboring exposure time. **t_p** for mvBlueFOX3-2032, mvBlueFOX3-2051, and mvBlueFOX3-2089 is **86.5us**, **102us** and **229.5us** respectively.*

*[3] **Block rate (kHz) @ n lines/block (kHz) = 1s / (readout time for n lines + readout time for vertical blank lines)**. For mvBlueFOX3-2032, mvBlueFOX3-2051, and mvBlueFOX3-2089 is **readout time for vertical blank lines 190us**, **220us** and **509us** respectively.*

3.7 Advantages of block scan mode over line scan camera

1. Standard interface: USB3 Vision and GigE Vision (in an upcoming firmware version) instead of CoaXPress and Cameralink.
2. Easier system setup: since the camera can use area scan too, it is much easier to adjust the focus and get a sharp image than line scan camera.
3. Less blocks loss due to FPGA in camera.
4. Less load on the host: blocks are collected in the camera.
5. Less pricy than line scan camera (with the same line rate).