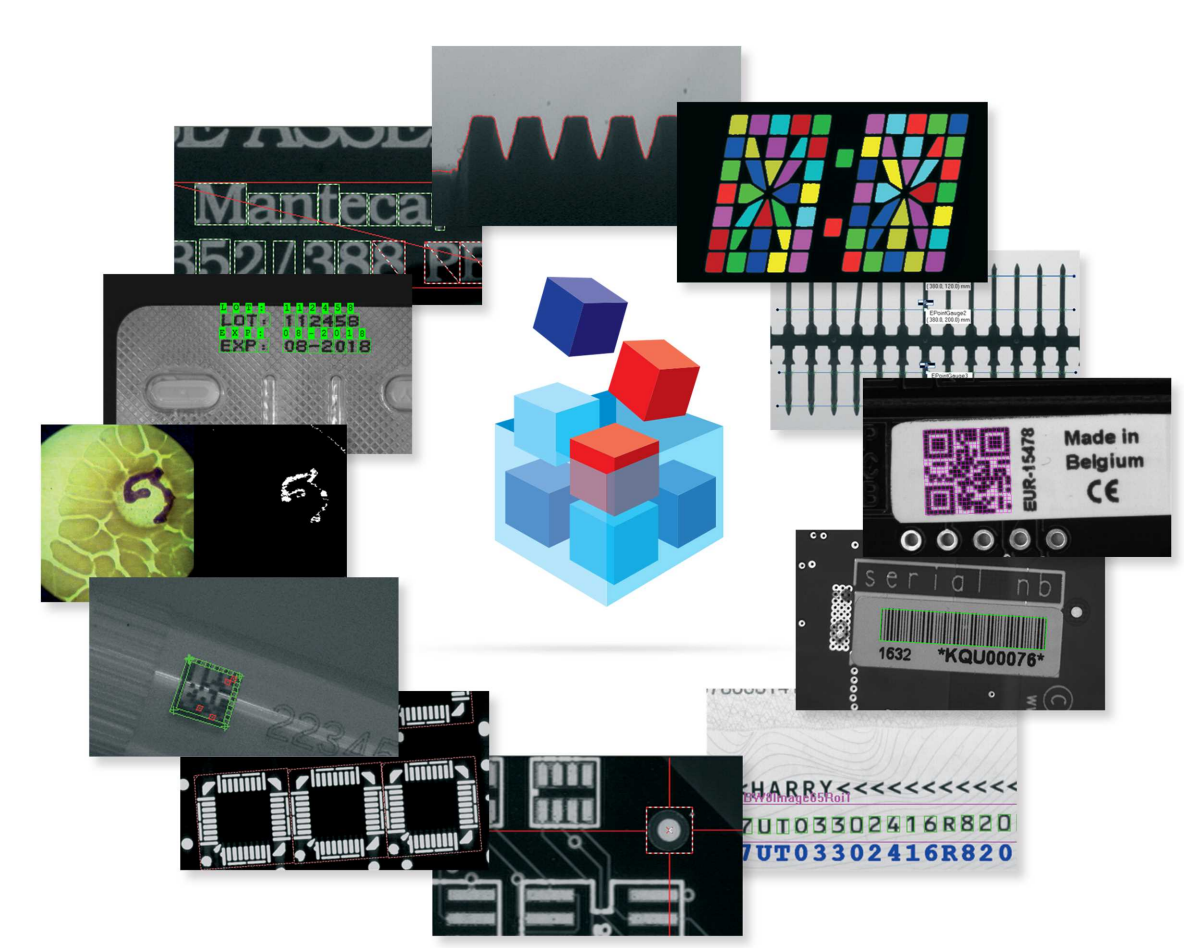
EURESYS™
Excellence in vision

사용자 설명서

Open eVision

이미지 전처리 라이브러리



© EURESYS s.a.0000 - Document D121KO-Using Image Pre-Processing Libraries C++-Open eVision-2.7.1.1114 built on 2019-03-07

사용 약관

EURESYS s.a. 는 EURESYS s.a.의 하드웨어 및 소프트웨어의 부속 문서, 상표의 모든 재산권, 소유권, 이권을 보유합니다.

이 설명서에 언급된 회사 및 제품의 모든 이름은 해당 소유자의 상표일 수 있습니다.

이 문서에 포함된 EURESYS s.a.의 자료, 하드웨어 또는 소프트웨어, 브랜드를 사전 통지 없이 라이선싱, 사용, 임대, 임차, 번역, 재현, 복사 또는 수정하는 행위는 허용되지 않습니다.

EURESYS s.a. 는 언제든 자사 재량에 따라 사전 통지 없이 제품 사양을 수정하거나 이 문서에서 제공하는 정보를 변경할 수 있습니다.

EURESYS s.a. 는 EURESYS s.a.의 하드웨어 도는 소프트웨어 사용과 관련하여 발생되는 일체의 매출, 수익, 영업권, 데이터, 정보 시스템의 손실 또는 피해 또는 기타 특별하거나, 우발적이거나, 간접적이거나, 필연적인 또는 징벌적인 손해에 대해 책임을 지지 않으며, 이는 본 문서의 누락 또는 오류로 인한 결과일 경우에도 마찬가지입니다.

이 문서는 Open eVision 2.7.1의 부속 자료입니다(문서 빌드 1114).
© 0000 EURESYS s.a.

목차

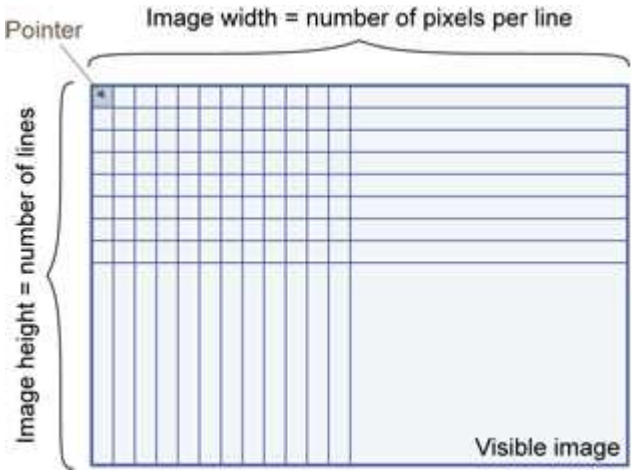
1. 픽셀 컨테이너 및 파일 조작하기

1.1. 픽셀 컨테이너 정의

이미지

Open eVision 이미지 개체에는 직사각형 이미지를 표현하는 이미지 데이터가 포함됩니다.

각 이미지 개체에는 포인터를 통해 액세스 가능하며 픽셀 값이 행 단위로 연속 저장되는 데이터 버퍼가 있습니다.



이미지 주요 매개변수

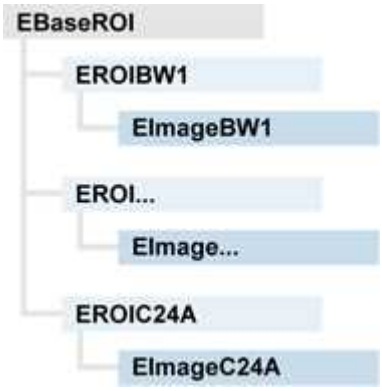
Open eVision 이미지 개체에는 `EBaseROI` 매개변수로 특성화되는 직사각형 픽셀의 배열이 있습니다.

- `Width(너비)`는 이미지 행당 열(픽셀)의 수입니다.
- `Height(높이)`는 이미지 행의 수입니다. (최대 너비/높이는 Open eVision 32비트의 경우 32,767(2^{15}-1), Open eVision 64비트의 경우 2,147,483,647(2^{31}-1)입니다)
- `Size(크기)`는 너비와 높이입니다.

`Plane(평면)` 매개변수에는 컬러 성분의 수가 수록됩니다. 회색조 이미지 = 1. 컬러 이미지 = 3.

클래스

이미지 및 ROI 클래스는 추상 클래스 `EBaseROI`에서 유래되며 모든 속성을 상속합니다.



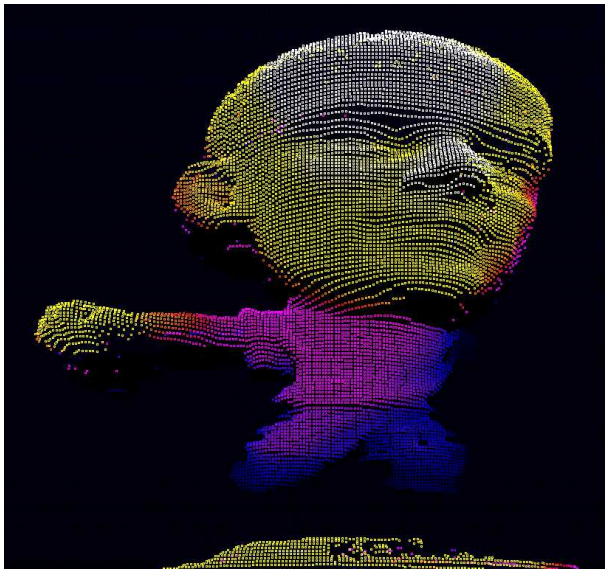
깊이 맵

깊이 맵은 3D 그레이 스케일 이미지를 사용하여 3D 오브젝트를 표현하는 방법으로 이미지의 각 픽셀은 3D 포인트를 나타냅니다.

픽셀 좌표는 점의 X 및 Y좌표의 표현이며 픽셀의 회색 음영 값은 점의 Z좌표 표현입니다

포인트 클라우드

포인트 클라우드(https://en.wikipedia.org/wiki/Point_cloud)는객체 표면의 불연속 위치를 나타내는 비정형 3D 점의 집합입니다.



3D 포인트 클라우드는 레이저 삼각 측량, 비행 시간 또는 구조 조명과 같은 다양한 3D 스캐닝 기법에 의해 생성됩니다.

1.2. 픽셀 컨테이너 유형

이미지

검정색, 흰색, 회색조, 컬러 등의 픽셀 유형에 따라 몇 가지 이미지 유형이 지원됩니다.

Easy.GetBestMatchingImageType 은 디스크에 있는 지정된 파일에 대해 가장 잘 일치하는 이미지 유형을 반환합니다.

BW1	1비트 흑백 이미지(8픽셀이 1바이트에 저장됨)	EImageBW1
BW8	8비트 그레이스케일 이미지(각 픽셀이 1바이트에 저장됨)	EImageBW8
BW16	16비트 그레이스케일 이미지(각 픽셀이 2바이트에 저장됨)	EImageBW16
BW32	32비트 그레이스케일 이미지(각 픽셀이 4바이트에 저장됨)	EImageBW32
C15	15비트 컬러 이미지(각 픽셀이 2바이트에 저장됨) Microsoft® Windows RGB15 컬러 이미지 및 MultiCam RGB15 형식과 호환됩니다.	EImageC15

C16	16비트 컬러 이미지(각 픽셀이 2바이트에 저장됨) Microsoft® Windows RGB16 컬러 이미지 및 MultiCam RGB16 형식과 호환됩니다.	EImageC16
C24	C24 이미지는 24비트 컬러 이미지를 저장합니다(각 픽셀이 3바이트에 저장됨). Microsoft® Windows RGB24 컬러 이미지 및 MultiCam RGB24 형식과 호환됩니다.	EImageC24
C24A	C24A 이미지는 32비트 컬러 이미지를 저장합니다(각 픽셀이 4바이트에 저장됨). Microsoft® Windows RGB32 컬러 이미지 및 MultiCam RGB32 형식과 호환됩니다.	EImageC24A

깊이 맵

8비트 및 16비트 깊이 맵 값은 2D Open eVision 이미지와 호환되는 버퍼에 저장됩니다.

EDepth8	8비트 깊이 맵(각 픽셀이 정수로 1바이트에 저장됨)	EDepthMap8
EDepth16	16비트 깊이 맵(각 픽셀이 고정점으로 2바이트에 저장됨)	EDepthMap16
EDepth32f	32비트 깊이 맵(각 픽셀이 부동 소수로 4바이트에 저장됨)	EDepthMap32f

포인트 클라우드

포인트 클라우드	점 좌표 세트(부동 소수로저장됨)	E3DPointCloud

1.3. 지원되는 이미지 파일 유형

유형	설명
BMP	비압축 이미지 데이터 형식(Windows 비트맵 형식)
JPEG	Joint Photographic Expert Group에서 발표한 데이터 손실 압축 표준이며 ISO/IEC 10918-1로 등록되었습니다. 압축을 적용하면 복구 불가능하게 품질이 손실됩니다.
JFIF	JPEG File Interchange Format

유형	설명
JPEG-2000	Joint Photographic Expert Group에서 발표한 데이터 손실 압축 표준이며 ISO/IEC 15444-1 및 ISO/IEC 15444-2로 등록되었습니다. Open eVision은 손실 압축 형식, 파일 형식, 코드 스트림 변종만을 지원합니다. - **코드 스트림**에 이미지 샘플이 설명됩니다. - **파일 형식**에는 이미지 해상도 및 색 공간과 같은 메타 정보가 포함됩니다.
PNG	무손실 데이터 압축 방식(Portable Network Graphics).
직렬화됨	Open eVision 이미지 개체의 **직렬화**를 통해 얻은 Euresys 독자 이미지 파일 형식.
TIFF	태그 이미지 파일 형식은 현재 Adobe Systems에서 관리하며, LibTIFF 서드 파티 라이브러리를 통해 TIFF 5.0 또는 6.0 TIFF 사양으로 작성된 이미지를 처리할 수 있습니다. 파일 **저장** 작업은 무손실이며, 1비트 바이너리 픽셀 유형에는 CCITT 1D 압축이, 다른 모든 유형에는 LZW 압축이 사용됩니다. 파일 **로드** 작업은 LibTIFF 사양에 열거된 모든 TIFF 버전을 지원합니다.

1.4. 픽셀 및 파일 형식 호환성

깊이 맵에서 이미지 변환

8비트 및 16비트 깊이 맵의 경우 `AsImage()` 메서드는 Open eVision'의 2D 처리 기능과 함께 사용할 수 있는 호환 가능한 이미지 객체 (각각 `EImageBW8` 및 `EImageBW16`) 를 반환합니다.

픽셀 및 파일 형식 호환성

픽셀 액세스

픽셀을 액세스하는 데 권장되는 방법은 `SetImagePtr` 및 `GetImagePtr`을 사용하여 사용자 자신의 코드에 이미지 버퍼 액세스를 임베드하는 것입니다. 또한 이미지 구조 및 메모리 할당 및 픽셀 값 검색을 참조하십시오.

다음 방법은 각 함수 호출에 따라 발생하는 오버헤드로 인해 사용이 제한적입니다.

직접 액세스

`EROIBW8::GetPixel` 및 `SetPixel` 메서드는 모든 이미지와 ROI 클래스에 구현되어 있으며, 지정된 좌표에서 픽셀 값을 읽거나 쓸 수 있습니다. 이미지의 모든 픽셀을 스캔하려면, X 좌표와 Y 좌표에서 이중 루프를 실행하고 각 반복에 `GetPixel` 또는 `SetPixel`을 사용할 수 있지만, 이는 권장되지 않습니다.

성능상의 이유로 많은 수의 픽셀을 처리해야 하는 경우에는 이러한 접근자를 사용하지 않아야 합니다. 이러한 경우 `GetBufferPtr()`를 사용하여 내부 버퍼 포인트를 검색하고 포인터를 반복하는 것이 좋습니다.

BW8 픽셀에 대한 간편 액세스

BW8 이미지에서, `EBW8PixelAccessor::GetPixel` 또는 `SetPixel`을 호출하는 것이 직접 `EROIBW8::GetPixel` 또는 `SetPixel`을 사용하는 것보다 빠릅니다.

지원되는 구조

- `EBW1`, `EBW8`, `EBW32`
- `EC15` (*), `EC16` (*), `EC24` (*)
- `EC24A`
- `EDepth8`, `EDepth16`, `EDepth32f`,

(*) 이 형식은 RGB15(5-5-5 비트 패킹), RGB16(5-6-5 비트 패킹), RGB32(RGB + 알파 채널)를 지원하지만, 일체의 작업 전에 `EasyImage::Convert`를 사용하여 EC24로/에서 변환해야 합니다.

> **참고:** *eVision 6.5 이전 버전과의 변환도 원활할 것이며, 이미지 픽셀 유형은 정수형의 typedef를 사용하여 정의되고, 픽셀 값은 무부호화 수로 처리되며, 이전 유형으로/에서 묵시적 변환이 제공됩니다.*

로드 또는 저장 작업 중의 픽셀 및 파일 유형 호환성

유형	BMP	JPEG	JPEG2000	PNG	TIFF	직렬화됨
BW1	Ok	N/A	N/A	Ok	Ok	Ok
BW8	Ok	Ok	Ok	Ok	Ok	Ok
BW16	N/A	N/A	Ok	Ok	Ok (***)	Ok
BW32	N/A	N/A	N/A	N/A	Ok (***)	Ok
C15	Ok	Ok (**)	Ok (**)	Ok (**)	Ok (**)	Ok
C16	Ok	Ok (**)	Ok (**)	Ok (**)	Ok (**)	Ok
C24	Ok	Ok	Ok	Ok	Ok (**)	Ok
C24A	Ok	N/A	N/A	Ok	N/A	Ok
Depth8	Ok	Ok	Ok	Ok	Ok	Ok
Depth16	N/A	N/A	Ok	Ok	Ok (***)	Ok
Depth32f	N/A	N/A	N/A	N/A	N/A	Ok

N/A: 지원되지 않습니다. 해당 조합을 사용하면 예외가 발생합니다.

Ok: 데이터 손실 없이 이미지 무결성이 보존됩니다(JPEG 및 JPEG2000, 손실 압축 제외).

(**) C15 및 C16 형식은 저장 작업 도중 자동으로 C24로 변환됩니다.

(***) BW16 및 BW32는 베이스라인 TIFF 리더에서 지원하지 않습니다.

1.5. 컬러 유형

EISH: 강도, 채도, 색조 컬러 시스템.

ELAB: CIE 명도, a*, b* 컬러 시스템.

ELCH: 명도, 크로마, 색조 컬러 시스템.

ELSH: 명도, 채도, 색조 컬러 시스템.

ELUV: CIE 명도, u*, v* 컬러 시스템.

ERGB: NTSC/PAL/SMPTE RGB(적, 녹, 청) 컬러 시스템.

EVSH: 값, 채도, 색조 컬러 시스템.

EXYZ: CIE XYZ 컬러 시스템.

EYIQ: CCIR 루마, 인페이즈, 쿼드러처 컬러 시스템.

EYSH: CCIR 루마, 채도, 색조 컬러 시스템.

EYUV: CCIR 루마, U 크로마, V 크로마 컬러 시스템.

2. 픽셀 컨테이너 및 파일 조작하기

2.1. 픽셀 컨테이터 파일 저장

이미지 및 깊이 맵

이미지의 `Save` 메서드 나 깊이 맵 또는 Z맵의 `SaveImage` 메서드는 이미지 또는 깊이 맵 또는 Z맵 객체의 이미지 데이터를 두 개의 인수를 사용하여 파일에 저장합니다.

- 경로: 경로, 파일 이름, 파일 이름 확장명.
- 이미지 파일 유형. 누락된 경우, 파일 이름 확장명이 사용됩니다.

65,536(너비 또는 높이 중 하나)보다 큰 이미지는 Open eVision 독자 형식으로 저장해야 합니다.

다음과 같은 경우 저장할 때 예외가 발생합니다.

- 요청된 이미지 파일 형식이 이미지 픽셀 유형과 호환되지 않을 경우
- 자동 파일 유형 선택 방식과 파일 이름 확장명이 지원되지 않을 경우

16비트 깊이 맵을 저장할 때 고정 소수점 정밀도가 손실되고 픽셀은 16비트 정수로 간주됩니다.

이미지 파일 유형 인수

인수	이미지 파일 유형
EImageFileType_Auto(*)	파일 이름 확장명에 의해 자동으로 결정됩니다. 아래를 참조하십시오.
EImageFileType_Euresys	Open eVision 직렬화
EImageFileType_Bmp	Windows 비트맵 - BMP
EImageFileType_Jpeg	JPEG File Interchange Format - JFIF
EImageFileType_Jpeg2000	JPEG 2000 파일 형식/코드 스트림 - JPEG2000
EImageFileType_Png	Portable Network Graphics - PNG
EImageFileType_Tiff	Tagged Image File Format - TIFF

(*) 기본값

인수가 ImageFileType_Auto이거나 누락된 경우 할당되는 이미지 파일 유형

파일 이름 확장명(*)	자동으로 할당되는 이미지 파일 유형
BMP	Windows 비트맵 형식
JPEG, JPG	JPEG File Interchange Format - JFIF
JP2	JPEG 2000 파일 형식
J2K, J2C	JPEG 2000 코드 스트림
PNG	Portable Network Graphics
TIFF, TIF	Tagged Image File Format

(*) 대소문자 구분 없음.

JPEG 및 JPEG2000 손실 압축 방식으로 저장하기

SaveJpeg 및 SaveJpeg2K의 경우 압축 이미지를 저장할 때 압축 품질을 지정할 수 있습니다. 여기에는 두 가지 인수가 있습니다.

- 경로: 경로, 파일 이름, 파일 이름 확장명이 포함된 문자열.
- 이미지 파일의 압축 품질, [0 ~ 100] 범위의 정수 값.
  SaveJpeg는 JPEG File Interchange Format – JFIF를 사용하여 이미지 데이터를 저장합니다.
  SaveJpeg2K는 JPEG 2000 파일 형식을 사용하여 이미지 데이터를 저장합니다.

JPEG 압축 값

JPEG 압축	설명
JPEG_DEFAULT_QUALITY (-1)	기본 품질(*)
100	최고의 이미지 품질, 가장 낮은 압축 비율
75	좋은 이미지 품질(*)
50	보통 이미지 품질
25	평균 이미지 품질
10	나쁜 이미지 품질

(*) 좋은 이미지 품질(75)에 해당하는 기본 품질.

대표적인 JPEG 2000 압축 품질 값

JPEG 2000 압축	설명
-1	기본 품질(*)
1	최고의 이미지 품질, 가장 낮은 압축 비율

JPEG 2000 압축	설명
16	좋은 이미지 품질(*) (16:1 비율)
512	최저 이미지 품질, 가장 높은 압축 비율

(*) 좋은 이미지 품질(16:1 비율)에 해당하는 기본 품질.

포인트 클라우드

- Save메서드를 사용하여 포인트 클라우드를 Open eVision 독점 파일 형식으로 저장합니다.
- SavePCD 메서드를 사용하여 PCL(포인트 클라우드 라이브러리)과 같은 다른 소프트웨어와 호환되는 ASCII 또는 이진 파일에 포인트 클라우드를 저장합니다.

PCD 형식은 ASCII 및 이진 모드에서 지원됩니다.

2.2. 픽셀 컨테이너 파일 로드

이미지 및 깊이 맵

- Load(로드) 메서드는 이미지 개체에 이미지 데이터를 로드합니다.
  - 여기에는 하나의 인수 **path:** 경로, 파일 이름, 파일 이름 확장명이 있습니다.
  - 파일 유형은 파일 형식에 따라 결정됩니다.
  - 대상 이미지는 디스크에 있는 이미지의 크기에 따라 자동으로 크기가 조정됩니다.
- 다음과 같은 경우 Load 메서드에서 예외가 발생합니다.
  - 파일 유형 인식 실패
  - 파일 유형이 이미지 개체의 픽셀 유형과 호환되지 않습니다.

Open eVision 1.1 이상의 직렬화된 이미지 파일은 이전 버전 Open eVision의 직렬화된 이미지 파일과 호환되지 않습니다.

깊이 맵에서 BW16 이미지(정수 값 포함)를 로드할 때 깊이 맵(기본값 0)에 설정된 고정 소수점 정밀도는 변경되지 않고 그대로 사용됩니다.

포인트 클라우드

- Load 메서드를 사용하여 포인트 클라우드를 Open eVision 독점 파일 형식으로 저장합니다.
- LoadPCD 메서드를 사용하여 PCL(포인트 클라우드 라이브러리)과 같은 다른 소프트웨어와 호환되는 ASCII 또는 이진 파일에 포인트 클라우드를 저장합니다.

2.3. 메모리 할당

내장 또는 외장 메모리 할당을 통해 이미지를 구성할 수 있습니다.

내장 메모리 할당

이미지 개체가 동적으로 버퍼를 할당 및 해제합니다. 메모리가 투명하게 관리됩니다.
이미지 크기가 변화하면 재할당이 일어납니다.
이미지 개체가 소멸되면 버퍼의 할당이 해제됩니다.
이미지에 내장 메모리 할당을 지정하려면:

1. 너비 및 높이 인수를 사용하여 또는 `SetSize` 함수를 사용하여 예를 들면 `EImageBW8` 형식의 이미지 개체를 생성하십시오.
2. 지정된 픽셀을 액세스하십시오. 이는 다음 몇 가지 함수를 통해 수행할 수 있습니다. `GetImagePtr`은 지정된 좌표에서 픽셀의 첫 바이트에 대한 포인터를 반환합니다.

외장 메모리 할당

사용자가 버퍼 할당을 제어하거나, 메모리 버퍼에 있는 서드 파티 이미지를 Open eVision 이미지에 연결할 수 있습니다.
이미지 크기와 버퍼 주소를 지정해야 합니다.
이미지 개체가 소멸되어도 버퍼에는 영향이 없습니다.
이미지에 외장 메모리 할당을 지정하려면:

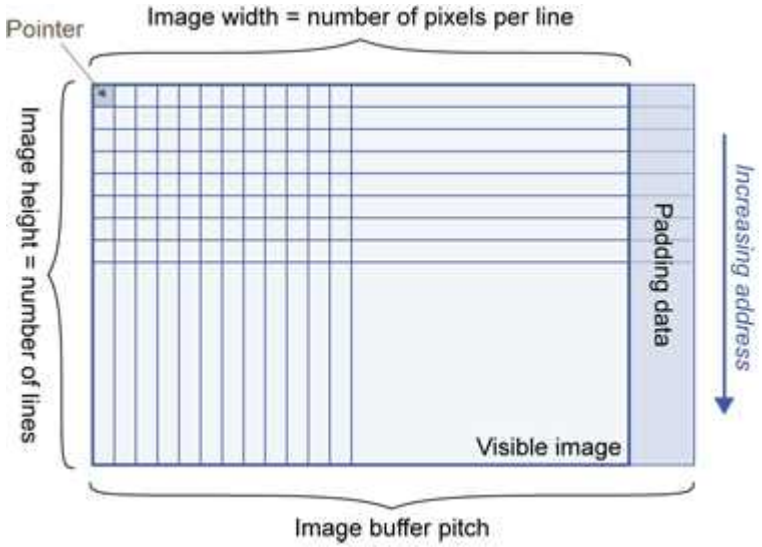
1. 예를 들면 `EImageBW8` 형식의 이미지 개체를 지정하십시오.
2. 적절한 크기의 정렬된 버퍼를 만드십시오(이미지 버퍼 참조).
3. `SetSize` 함수를 사용하여 이미지 크기를 설정하십시오.
4. `GetImagePtr`을 사용하여 버퍼를 액세스하십시오. 픽셀 값 가져오기를 참조하십시오.

2.4. 이미지 및 깊이 맵 버퍼

이미지 및 깊이 맵 픽셀은 위쪽에서 아래쪽으로, 왼쪽에서 오른쪽으로 Windows 비트맵 형식(하향식 DIB[1])으로 연결된 버퍼에 연속으로 저장됩니다.

버퍼 주소는 이미지의 왼쪽 상단 픽셀이 포함된 버퍼의 시작 주소에 대한 포인터입니다.

[1]장치 독립적 비트맵



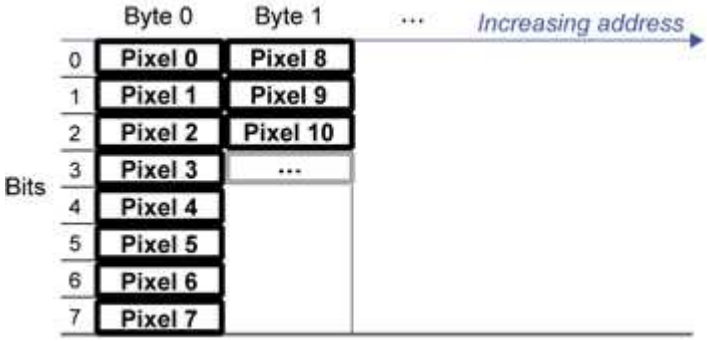
이미지 버퍼 피치

- 정렬은 4바이트의 배수로 이루어져야 합니다.
- Open eVision 1.2 이상에서는 성능상의 이유로 기본 피치가 32바이트입니다(Open eVision 1.1.5의 경우 8바이트).

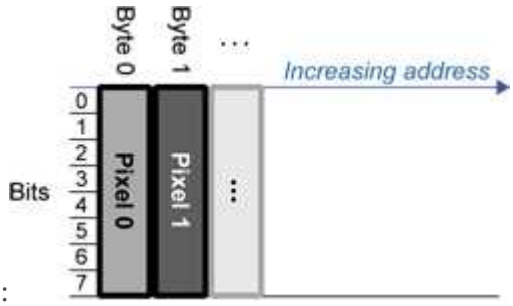
메모리 레이아웃

- `EImageBW1`은 한 바이트에 8픽셀을 저장합니다.

  BW1 이미지 버퍼의 첫 두 픽셀의 메모리 레이아웃 예:



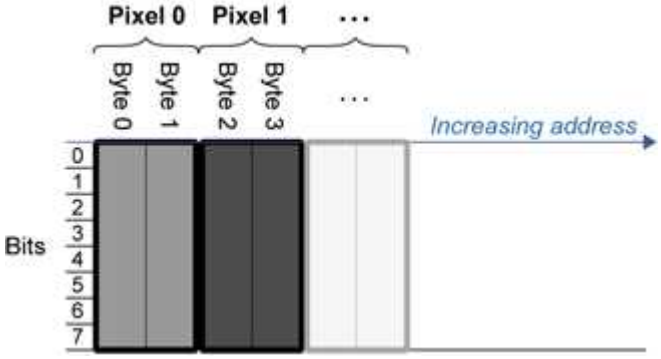
- `EImageBW8` 및 `EDepthMap8`는 각 픽셀을 1바이트에 저장합니다.



  BW8 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:

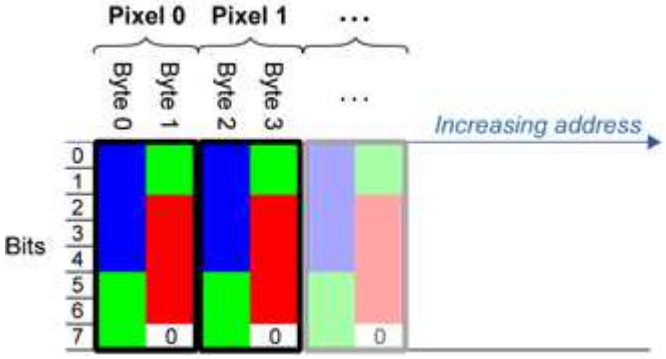
- `EImageBW16`은 각 픽셀을 16비트 워드(2 바이트)에 저장합니다.

BW16 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:



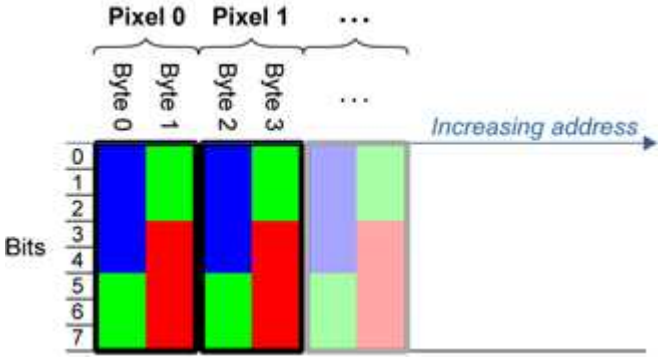
- EImageC15은 각 픽셀을 2바이트에 저장합니다. 각 컬러 성분은 5비트로 코드화됩니다. 16번째 비트는 미사용 상태로 유지됩니다.

  C15 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:



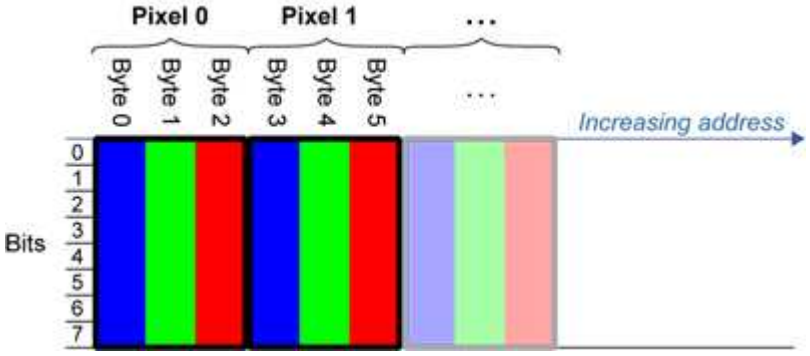
- EImageC16은 각 픽셀을 2바이트에 저장합니다. 첫 번째와 세 번째 컬러 성분은 5비트로 코드화됩니다.
  두 번째 컬러 성분은 6비트로 코드화됩니다.

  C16 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:



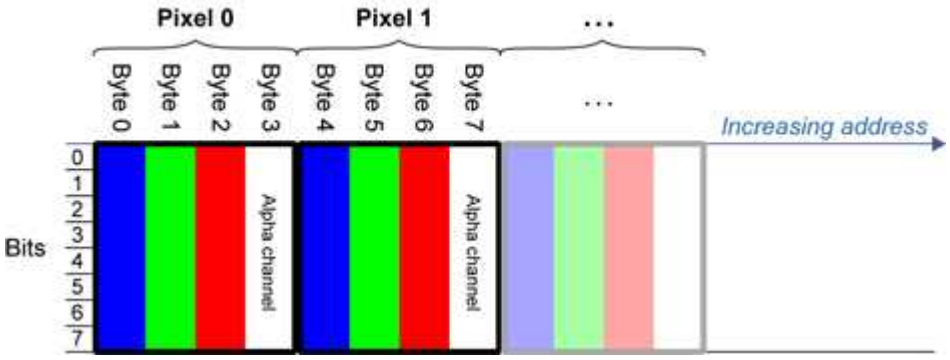
- EDepthMap16은 고정 소수점 형식을 사용하여 각 픽셀을 2바이트에 저장합니다.
- EImageC24는 각 픽셀을 3바이트에 저장합니다. 각 컬러 성분은 8비트로 코드화됩니다.

C24 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:



- `EImageC24A`는 각 픽셀을 4바이트에 저장합니다. 각 컬러 성분은 8비트로 코드화됩니다. 알파 채널 또한 8비트로 코드화됩니다.

  C24A 이미지 버퍼의 첫 픽셀의 메모리 레이아웃 예:



- `EDepthMap32f`은 각 픽셀을 플로트 형식을 사용하여 4바이트에 저장합니다.

2.5. 이미지 그리기 및 덮어쓰기

- 그리기는 Windows GDI[1] 시스템 호출을 사용합니다.
  MFC[2] 애플리케이션은 일반적으로 그리기에 장치 컨텍스트에 대한 포인터가 제공되는 `OnDraw` 이벤트 핸들러를 사용합니다.
  Borland/CodeGear OWL 또는 VCL에는 **Paint** 이벤트 핸들러가 사용됩니다.
- 256 컬러 표시 모드의 색상표가 최적의 렌더링을 제공합니다. 회색조 이미지는 LUT[3]를 사용하여 개선할 수 있습니다(히스토그램 확장 기법 또는 유사 색상 지정 활용).
- 줌은 수직 및 수평 방향으로 다를 수 있습니다.
- `DrawFrameWithCurrentPen` 메서드는 프레임을 그립니다.
- **비파괴적 오버레이** 그리기 연산은 `MoveTo/LineTo`와 같이 이미지 내용을 변경하지 않습니다.

---

[1]그래픽 장치 인터페이스

[2]Microsoft Foundation Class

[3]조회 테이블

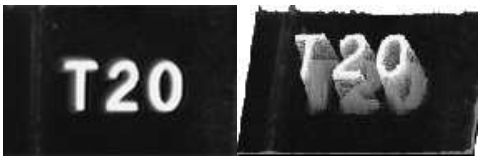
- **파괴적 오버레이** 그리기 연산은 `Easy::OpenImageGraphicContext`와 같이 이미지 내부에 그림으로써 이미지 내용을 변경합니다. 회색조 [컬러] 이미지는 회색조 [컬러] 오버레이만 받을 수 있습니다.

2.6. 2D 이미지의 3D 렌더링

X 축, 그리고 Y 축 주변으로 이미지를 회전시켜 볼 수 있습니다.

회색 3D 렌더링

`Easy::Render3D`는 회색조 값이 고도인 3차원 렌더링을 준비합니다.
확대 비율은 세 방향(X = 너비, Y = 높이, Z = 깊이)으로 지정할 수 있습니다.
렌더링된 이미지는 독립적인 점으로 표시되며, 그 크기를 조정하여 표면의 불투명도를 바꿀 수 있습니다.

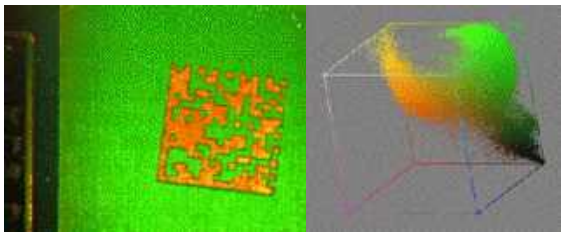


3D 렌더링

컬러 히스토그램 3D 렌더링

`Easy::RenderColorHistogram`은 컬러 이미지 히스토그램의 3차원 렌더링을 준비합니다.
RGB 값의 군집화와 분산을 보여줄 수 있도록 RGB 공간(XY 평면이 아님)에 픽셀이 그려집니다.
이 기능으로 다른 컬러 시스템(EasyColor를 사용하여 변환)의 픽셀을 처리할 수도 있지만, 픽셀을 본래 컬러로 표시하려면 원시(raw) RGB 이미지가 필요합니다.

확대 비율은 세 방향(X = 너비, Y = 높이, Z = 깊이) 모두 지정할 수 있습니다.

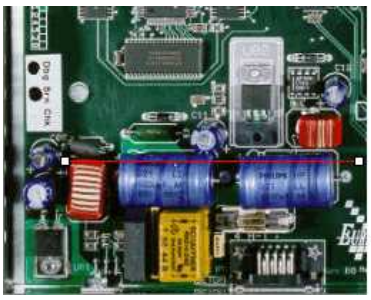


컬러 히스토그램 렌더링

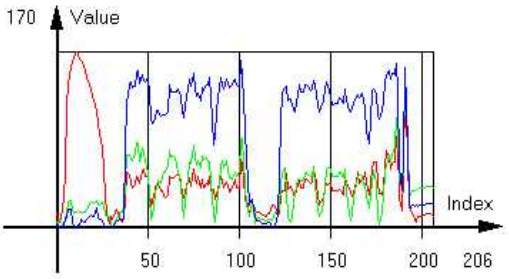
2.7. 벡터 유형 및 주요 속성

벡터는 픽셀의 일차원 배열(이미지 프로파일 또는 윤곽선에서 획득)입니다.

EVector는 모든 벡터의 기준 클래스입니다. 여기에는 주로 부품 계수용 또는 직렬화에 사용되는 유형이 지정되지 않은 모든 메서드가 포함됩니다.



C24 이미지 내의 프로파일



프로파일상의 RGB 값 그래프

Index	Red	Green	Blue
0	15	5	3
1	7	4	0
2	5	8	0
3	9	5	0
4	29	1	0
5	55	6	9
6	120	15	9
7	139	24	17
8	157	26	18
9	161	17	6
10	165	13	0
11	170	14	1

RGB 값 배열(EC24Vector)

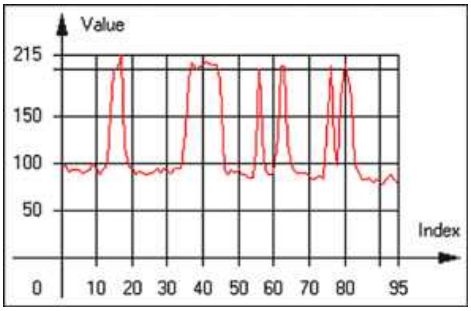
벡터는 요소의 배열을 관리합니다. 메모리 할당이 투명하므로, 벡터는 동적으로 크기를 조정할 수 있습니다. 함수에서 벡터를 사용할 때마다 벡터 유형, 크기, 구조가 해당 함수의 요구에 적합하게 자동으로 조정됩니다.

벡터의 사용법은 상당히 직관적입니다.

1. 벡터 생성자와 필요할 경우 사전 할당 요소를 사용하여 **적절한 유형의 벡터를 생성할 수 있습니다**.

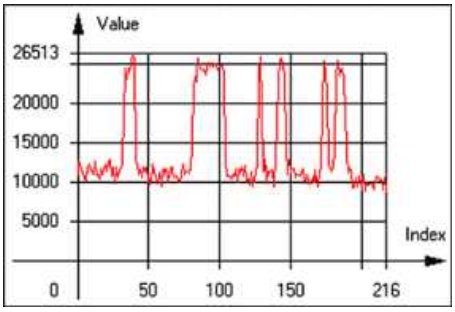
벡터 유형

- EBW8Vector: 회색조 픽셀 값의 시퀀스이며, 흔히 이미지 프로파일에서 추출됩니다 (EasyImage::Lut, EasyImage::SetupEqualize, EasyImage::ImageToLineSegment, EasyImage::LineSegmentToImage, EasyImage::ProfileDerivative 등에서 사용됨).



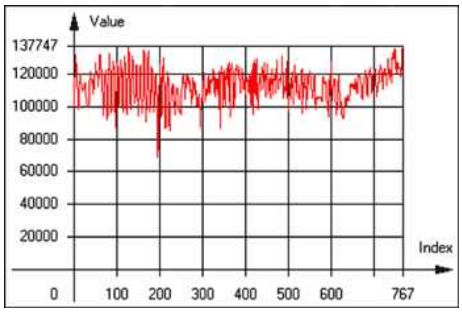
EBW8Vector의 그래픽 표현(Draw 메서드 참조)

- EBW16Vector: 확장 범위(16비트)를 사용하며 주로 중간 계산에 사용되는 회색조 픽셀 값의 시퀀스입니다.



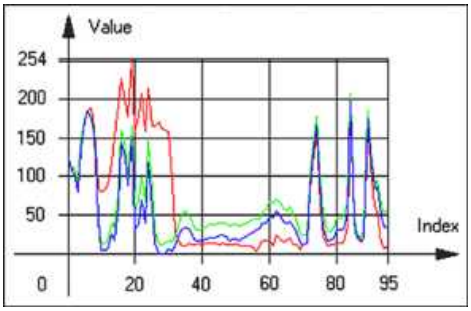
EBW16Vector의 그래픽 표현

- EBW32Vector: 확장 범위(32비트)를 사용하며 주로 중간 계산에 사용되는 회색조 픽셀 값의 시퀀스입니다.
  (EasyImage::ProjectOnARow, EasyImage::ProjectOnAColumn 등에서 사용됨).



EBW32Vector의 그래픽 표현

- EC24Vector: 컬러 픽셀 값의 시퀀스이며, 흔히 이미지 프로파일에서 추출됩니다
  (EasyImage::ImageToLineSegment, EasyImage::LineSegmentToImage, EasyImage::ProfileDerivative 등에서 사용됨).

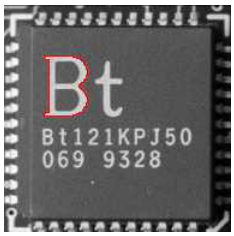


EC24Vector의 그래픽 표현

- EBW8PathVector: 이미지 프로파일 또는 윤곽선에서 해당 픽셀 좌표와 함께 추출되는 회색조 픽셀 값의 시퀀스입니다.
  (EasyImage::ImageToPath, EasyImage::PathToImage 등에서 사용됨).

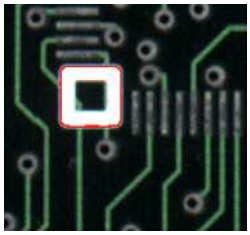
EBW8PathVector의 그래픽 표현(Draw 메서드 참조)

- EBW16PathVector: 이미지 프로파일 또는 윤곽선에서 해당 픽셀 좌표와 함께 추출되는 회색조 픽셀 값의 시퀀스입니다.
  (EasyImage::ImageToPath, EasyImage::PathToImage 등에서 사용됨).



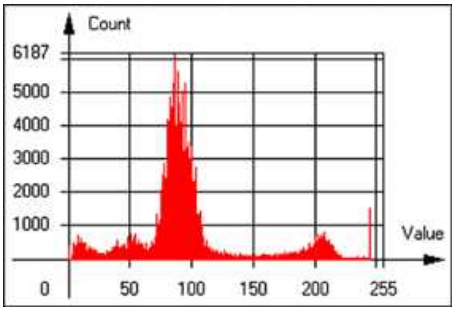
EBW16PathVector의 그래픽 표현(Draw 메서드 참조)

- EC24PathVector: 이미지 프로파일 또는 윤곽선에서 해당 픽셀 좌표와 함께 추출되는 컬러 픽셀 값의 시퀀스입니다.
  (EasyImage::ImageToPath, EasyImage::PathToImage 등에서 사용됨).



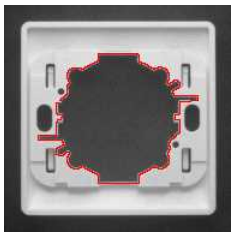
EC24PathVector의 그래픽 표현(Draw 메서드 참조)

- EBWHistogramVector: BW8 또는 BW16 이미지 내 픽셀의 빈도 카운트의 시퀀스입니다
  (EasyImage::IsodataThreshold, EasyImage::Histogram,
  EasyImage::AnalyseHistogram, EasyImage::SetupEqualize 등에서 사용됨).



EBWHistogramVector의 그래픽 표현(Draw 메서드 참조)

- EPathVector: 픽셀 좌표의 시퀀스입니다. 해당 픽셀은 연속적일 필요가 없습니다 (EasyImage::PathToImage 및 EasyImage::Contour에서 사용됨).



EPathVector의 그래픽 표현(Draw 메서드 참조)

- EPeakVector: 이미지 프로파일에서 발견된 피크입니다 (EasyImage::GetProfilePeaks에서 사용됨).
- EColorVector: 컬러의 명세입니다 (EasyColor::ClassAverages 및 EasyColor::ClassVariances에서 사용됨).

2. **벡터에 값을 채우십시오**. 먼저 EVector::Empty 멤버를 사용하여 벡터를 비우고, EC24Vector::AddElement 멤버를 호출하여 한 번에 하나씩 요소를 추가하십시오. 색인을 활용하여 어떤 요소든 액세스할 수 있습니다.
3. 읽기 또는 쓰기 어느 쪽이든 **벡터 요소를 액세스하십시오** 예를 들어 EC24Vector::operator[]와 같은 괄호 연산자를 사용하십시오.
4. **멤버** EVector::NumElements를 사용하여 현재 요소 수를 판정하십시오.
5. **벡터를 그리십시오**.
   픽셀 벡터는 요소 색인의 함수 형태인 요소 값의 그래프이며, 따라서 그 유형에 따라 그래픽 표시 모양이 결정됩니다. 벡터를 창 안에 그릴 수 있습니다. 가독성을 고려할 때 벡터 그림은 중성 배경에 표시되어야 합니다.
   그리기는 원하는 창에 연결된 장치 컨텍스트에 맞게 실행됩니다. 기본적으로, 곡선은 파란색으로, 주석은 검정색으로 표시됩니다. 정의할 수 있는 매개변수는 다음과 같습니다. graphicContext, width, height, origin, origin, color0, color1, color2.
   EC24Vector에는 곡선 하나가 아니라 각각 컬러 성분에 대응하는 곡선 세 개가 그려집니다. 기본적으로, 빨간색, 파란색, 녹색 펜이 사용됩니다.

2.8. ROI 주요 속성

ROI는 너비, 높이, 및 **원점** x **및** y **좌표**로 정의됩니다.
원점은 부모 이미지 또는 ROI의 왼쪽 상단 코너에 비례하여 지정됩니다.
ROI는 그 부모 이미지에 완전히 포함되어야 합니다.
`OrgX` 및 `Width(너비)`가 8의 배수일 경우 BW1 ROI의 처리/분석 속도가 훨씬 빨라집니다.

저장 및 로드

ROI를 별도의 이미지로 저장 또는 로드하여 마치 전체 이미지인 것처럼 사용할 수 있습니다. ROI는 전혀 **메모리 할당을 수행하지 않고** 절대 부모 이미지의 일부를 복제하지 않으며, 부모 이미지가 그 이미지 데이터에 대한 액세스를 제공합니다.

새로운 파일의 이미지 크기는 그 안에 로드되는 ROI의 크기와 일치해야 합니다. ROI 주변의 이미지는 변경되지 않습니다.

ROI 클래스

Open eVision ROI는 추상 클래스 `EBaseROI`에서 매개변수를 상속합니다.

픽셀 유형에 따라 몇 가지 ROI 유형이 있습니다. ROI는 해당 이미지 유형과 동일한 특성을 가집니다.

- `EROIBW1`
- `EROIBW8`
- `EROIBW16`
- `EROIBW32`
- `EROIC15`
- `EROIC16`
- `EROIC24`
- `EROIC24A`

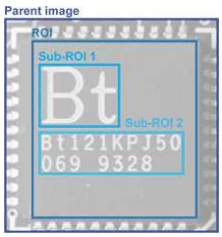
첨부

ROI는 부모, 위치, 크기를 설정하는 매개변수와 함께 부모(이미지/ROI)에 첨부되어야 하며, 해당 링크는 추종하는 포인터가 없도록 투명하게 업데이트됩니다.
일반 이미지는 다른 이미지 또는 ROI에 첨부할 수 없습니다.

중첩

Set 및 Get 함수는 직속 또는 최상위 부모 이미지와 관련하여 ROI의 너비, 높이, 원점 위치를 변경 또는 쿼리합니다.

이미지는 계층적인 방식으로 중첩될 수 있는 임의 수의 ROI를 수용할 수 있습니다. ROI를 이동하면 임베디드 ROI도 그에 따라 이동됩니다. The image/ROI classes provide several methods to traverse the hierarchy of ROIs associated with an image.



중첩된 ROI: 두 하위 ROI가 한 ROI에 첨부되었으며, 그 자체도 부모 이미지에 첨부됨

자르기

`CropToImage`는 이미지에서 부분적으로 벗어난 ROI를 잘라냅니다. 크기가 조정된 ROI는 절대 확장되지 않습니다.
어떤 함수에서 한계가 부모의 외부로 연장되는 ROI를 사용하려고 시도할 경우 예외가 발생합니다.

> **참고:** *(Open eVision 1.0.1 이하 버전에서는 ROI가 이미지 외부에 배치되고 때로 확장되는 경우, 조용히 크기가 조정되거나 재배치됩니다. ROI 한계가 부모 외부로 확장되면, 조용히 크기가 조정되어 부모 한계 내로 유지됩니다.)*

크기 조정 및 이동

- ROI는 다음 두 함수와 끌기 핸들을 사용하여 손쉽게 크기를 조정하거나 배치할 수 있습니다.
  - `EBaseROI::Drag`는 커서를 이동할 때 ROI 좌표를 조정합니다.
  - `EBaseROI::HitTest`는 커서가 드래그 핸들 위에 있을 때 알려줍니다. 핸들이 알려지면, `OnSetCursor` MFC 이벤트 핸들러를 통해 커서 형상이 변경될 수 있습니다. 끌기가 진행 중인 동안 HitTest를 호출하면 예측 불가능한 상태가 됩니다.
    HitTest를 OnSetCursor MFC 이벤트 핸들러에서 커서 형상을 바꾸는 데, 또는 OnLButtonDown과 같은 끌기 작업 전에 사용할 수 있습니다.
    (또는 Borland/CodeGear OWL에서 EvSetCursor 및 EvLButtonDown에)
    (또는 Borland/CodeGear VCL에서 FormMouseMove 및 FormMouseDown에).
    VB6에서 MouseDown, MouseMove, MouseUp 이벤트는 픽셀이 아니라 트윕 단위로 현재 커서 위치를 반환하므로, 변환이 필수입니다.

2.9. 임의적 ROI (ERegion)

참조:예: Inspecting Pads Using Regions / 코드 스니핏: ERegion

영역 또는 임의적 ROI

관심 영역(ROI)을 정의하고 사용하여 비전 도구로 처리되는 영역을 제한하고 처리 시간을 줄이고 최적화합니다.

Open eVision에서:

- **ROI** (`EROIxxx` 클래스)는 직사각형의 관심 영역을 지정합니다.
- **영역**(`ERegion` 클래스)은 임의의 모양의 ROI를 지정합니다. 영역을 사용하면 하나의 픽셀까지 이미지의 어느 부분이 처리에 사용되는지 정확하게 결정할 수 있습니다.

현재 다음 메소드 Open eVision만 `ERegions`를 지원합니다:

라이브러리	방법
EasyImage	EasyImage::Threshold
	EasyImage::DoubleThreshold
	EasyImage::Histogram
	EasyImage::Area
	EasyImage::AreaDoubleThreshold
	EasyImage::BinaryMoments
	EasyImage::WeightedMoments
	EasyImage::GravityCenter
	EasyImage::PixelCount
	EasyImage::PixelMax
	EasyImage::PixelMin
	EasyImage::PixelAverage
	EasyImage::PixelStat
	EasyImage::PixelVariance
	EasyImage::PixelStdDev
	EasyImage::PixelCompare
Easy3D	EDepthMapToMeshConverter::Convert
	EDepthMapToPointCloudConverter::Convert
	EStatistics::ComputePixelStatistics
	EStatistics::ComputeStatistics
EasyObject	EImageEncoder::Encode
EasyFind	EPatternFinder::Find

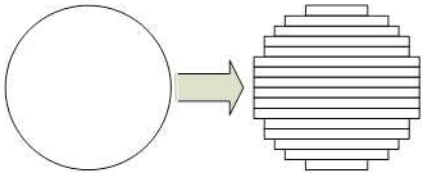
향후 Open eVision 배포에서는 `ERegions`의 지원이 모든 운영자에게 점차 확대될 것입니다.

영역 만들기

Open eVision는 필요한 모양에 따라 영역을 만드는 여러 가지 방법을 제공합니다.

ERegion은 모든 지역의 기본 클래스이며 가장 융통성이 있습니다. RLE (Run-Length Encoded) 표현을 사용하여 영역을 인코딩합니다.

- 영역의 RLE 표현은 런(가로 1픽셀 높이 슬라이스)으로 구성됩니다.
- 런은 횡좌표와 길이로 시작하는 세로 좌표 형식으로 저장됩니다.



원형 영역의 런-길이 인코딩

영역을 만들려면 다음 중 하나를 수행합니다:

- 기하학 기반 영역 클래스 중 하나를 사용합니다.
- EasyFind, EasyMatch 또는 EasyObject와 같은 다른 도구의 결과를 사용합니다.
- 다른 영역을 결합하거나 수정합니다.
- 마스크 이미지를 사용합니다.
- 직접 실행 목록을 제공합니다.

기하학 기반 영역

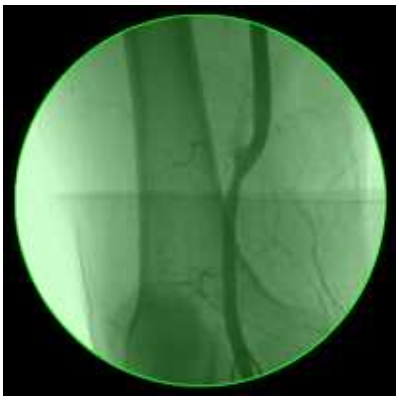
기하학 기반 영역은 단순한 기하학에 포함되는 특수화된 영역 클래스입니다. Open eVision는 현재 직사각형, 원, 타원 또는 다각형 기반의 클래스를 제공합니다.

이 클래스를 사용하여 기하학적 영역을 설정하고 변환, 회전 및 크기 조절을 사용하여 수정합니다. 변환 연산자는 소스 영역을 변경하지 않고 새로운 영역을 반환합니다.

- `ERectangleRegion`
  - `ERectangleRegion` 클래스의 윤곽선은 사각형입니다.
  - 중심, 폭, 높이 및 각도를 사용하여 정의합니다.
  - 또는 `ERectangleGauge` 인스턴스에서 반환한 것과 같은 `ERectangle` 인스턴스를 사용합니다.

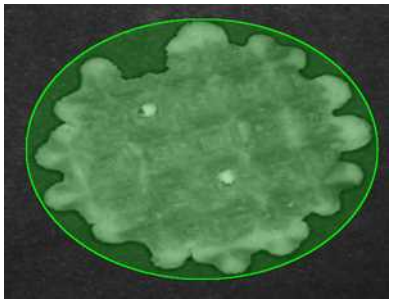
백그라운드에서 바코드를 분리하는 사각형 영역

- `ECircleRegion`
  - `ECircleRegion` 클래스의 윤곽선은 원입니다.
  - 중심과 반지름 또는 3개의 비 정렬점을 사용하여 정의합니다.
  - 또는 `ECircleGauge` 인스턴스에서 반환한 것과 같은 `ECircle` 인스턴스를 사용합니다.



X선 이미지의 유용한 부분을 포함하는 원형 영역

- EEllipseRegion
  - EEllipseRegion 클래스의 윤곽은 타원입니다.
  - 중심, 길이 및 반경 및 각도를 사용하여 정의합니다.



와플을 둘러싸는 타원 영역

- EPolygonRegion
  - EPolygonRegion 클래스의 윤곽선은 다각형입니다.
  - 정점 목록을 사용하여 생성됩니다.

키를 포함하는 다각형 영역

다른 도구의 결과 사용

ERegion 클래스는 다른 도구의 결과로 영역을 생성하는 전문 생성자 세트를 제공합니다. 도구 체인에서 이러한 생성자는 도구의 처리를 이전 도구에서 발행된 영역으로 제한합니다.

Open eVision는 다음 도구에 대한 생성자를 제공합니다:

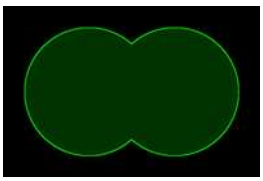
- EasyFind: `EFoundPattern`
- EasyMatch: `EMatchPosition`
- EasyGauge: `ECircle` 및 `ERectangle`
- EasyObject: `ECodedElement`

호환 가능하면 Open eVision는 기하학 기반 영역에 대한 특수 생성자도 제공합니다. 예를 들어, `ECircleRegion`은 `ECircle`을 사용하여 생성자를 제공합니다.

영역 결합

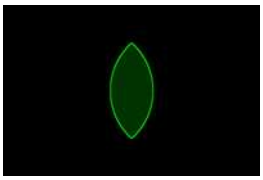
기존 영역을 결합하여 새 영역을 만들려면 다음 작업을 사용합니다:

- 조합
  - `ERegion::Union(const ERegion&, const ERegion&)` 메서드는 인수로 전달된 두 영역의 추가 영역을 반환합니다.



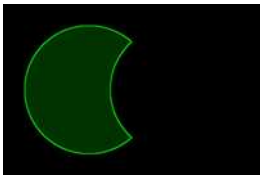
2개의 원 조합

- 교차
  - `ERegion::Intersection(const ERegion&, const ERegion&)` 메서드는 인수로 전달된 두 영역의 교차 영역인 영역을 반환합니다.



2개의 원 교차

- 제거
  - `ERegion::Substraction(const ERegion&, const ERegion&)` 메서드는 두 번째 인수를 제거한 후 인수로 전달한 첫 번째 영역을 반환합니다.



2개의 원 제거

영역 사용

영역을 지원하는 도구는 다음 규칙 중 하나를 따르는 메소드를 제공합니다:

- `Method(const EImage& source, const ERegion& region)`
- `Method(const EImage& source, const ERegion& region, EImage& destination)`

참고: *소스, 지역 및 대상은 호환 가능해야 합니다. 이는 해당 영역이 소스에 적어도 부분적으로 적합해야 하며 해당 소스와 대상의 크기가 동일해야함을 의미합니다.*

영역 준비하기

- Open eVision는 영역을 이미지에 적용할 때 자동으로 영역을 준비하지만 이 준비에는 시간이 걸릴 수 있습니다.
- 메서드에 대한 첫 번째 호출이 다음 호출보다 오래 걸리지 않으려면 적절한 `Prepare()` 메서드를 사용하여 미리 영역을 준비할 수 있습니다.
- 영역을 수동으로 준비하려면 이미지에 내부 RLE 설명을 적용합니다.

영역 그리기

`ERegion` 클래스는 영역을 표시하는 여러 가지 방법을 제공합니다:

- `ERegion::Draw()`는 제공된 장치 컨텍스트에서 반투명 방식으로 영역을 그립니다.
- `ERegion::DrawContour()`는 제공된 장치 컨텍스트에서 영역 윤곽선을 그립니다.
- `ERegion::ToImage()`는 영역을 제공된 대상 이미지에 마스크로 렌더링합니다.
  - 전경색과 배경색을 구성할 수 있습니다.
  - 너비와 높이로 이미지를 초기화했다면, Open eVision는 그 범위 내에서 영역을 렌더링합니다.
  - 그렇지 않으면, Open eVision는 전체 영역을 포함하도록 이미지의 크기를 조정합니다.
  - `ToImage()`를 사용하여 이를 지원하는 Open eVision함수의 마스크를 만듭니다.

ERegion 및 EROI

- 이전 EROI Open eVision클래스는 새로운 영역과 호환됩니다.
- 일부 도구는 EImage 대신 ERoi인 소스 및/또는 대상이 있는 영역을 사용할 수 있도록 다음 규칙 중 하나를 따릅니다:
  - Method(const ERoi& source, const ERegion& region)
  - Method(const ERoi& source, const ERegion& region, ERoi& destination)

이 경우 영역에 사용 된 좌표는 전체 이미지 공간 대신 축소된 ROI 공간에 상대적입니다.

ERegion 및 3D

- 새로운 영역은 Easy3D (EDepthMap 및 EZMap)의 2.5D 표현과 호환됩니다.
- 이러한 클래스를 사용할 때 처리 도메인을 줄일 수도 있습니다.

2.10. 플렉시블 마스크

ROI 대 플렉시블 마스크

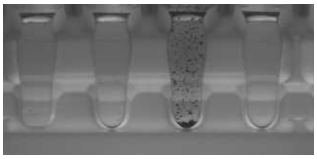
ROI 및 마스크는 이미지의 일부로 프로세싱을 제한합니다.

- 모든 Open eVision 함수에 "ROI 주요 속성" 페이지25이 적용됩니다. 관심 영역을 사용하면 픽셀 수를 줄여 처리 속도를 단축할 수 있습니다. Open eVision은 계층 구조적으로 중첩된 직사각형 ROI를 지원합니다.
- 플렉시블 마스크 단절된 ROI 또는 비정방형 형상을 처리하는 데에는 가 권장됩니다. 이는 일부 EasyObject 및 EasyImage 라이브러리 함수에서 지원됩니다.

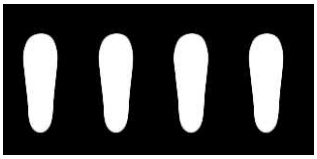
플렉시블 마스크

플렉시블 마스크는 소스 이미지와 동일한 높이와 너비의 BW8 이미지입니다. 여기에는 반드시 처리되어야 하는 영역의 형상과 무시 영역(처리 도중 고려되지 않을 부분)이 포함됩니다.

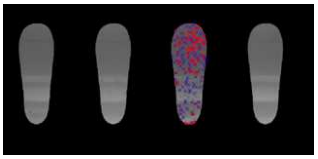
- 플렉시블 마스크에서 값이 0인 픽셀은 모두 무시 영역을 의미합니다.
- 플렉시블 마스크에서 값이 0이 아닌 픽셀은 모두 처리해야 하는 영역을 의미합니다.



소스 이미지



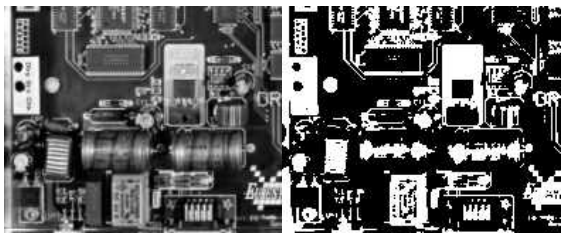
연결된 마스크



처리된 마스크 적용 이미지

플렉시블 마스크는 BW8 이미지를 출력할 수 있는 모든 애플리케이션과 일부 EasyObject 및 EasyImage 함수에서 만들 수 있습니다.

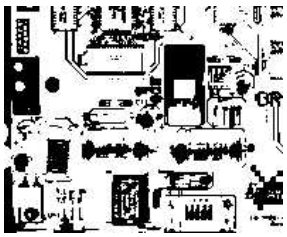
EasyImage에서 플렉시블 마스크 사용



소스 이미지(왼쪽)와 마스크 변수(오른쪽)

Easyimage에서 플렉시블 마스크를 사용하는 간단한 절차

1. **EasyImage에서 입력 마스크를 인수로 채택하는 함수를 호출하십시오**. 예를 들어, 흰색 레이어에 이어 검정색 레이어에 포함된 픽셀의 평균값을 평가할 수 있습니다.
2. **결과가 표시됩니다.**



결과 이미지

플렉시블 마스크를 지원하는 EasyImage 함수

- `EImageEncoder::Encode`에는 BW1, BW8, BW16, C24 소스 이미지에 대응하는 플렉시블 마스크 인수가 있습니다.
- AutoThreshold.
- 히스토그램 (`HistogramThreshold` 함수에는 마스크 인수를 포함한 오버로드가 없습니다).
- RmsNoise, SignalNoiseRatio.
- 오버레이(BW8 소스 이미지에는 마스크 인수를 포함한 오버로드 없음).
- ProjectOnAColumn, ProjectOnARow(벡터 프로젝션).
- ImageToLineSegment, ImageToPath(벡터 프로파일).

EasyObject에서 플렉시블 마스크 사용

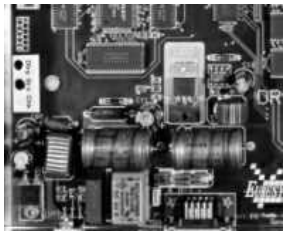
플렉시블 마스크는 BW8 이미지를 출력할 수 있는 모든 애플리케이션에서 또는 Open eVision 이미지 처리 함수를 사용하여 만들 수 있습니다.
EasyObject에서 플렉시블 마스크를 사용하여 이미지의 복잡한 영역 또는 형상이 단절된 영역으로 블롭 분석을 제한할 수 있습니다.

관심 개체가 이미지의 다른 영역과 동일한 회색조를 가졌다면, 플렉시블 마스크와 `Encode`(인코딩) 함수를 사용하여 "유지" 및 "무시" 영역을 정의할 수 있습니다.

플렉시블 마스크는 소스 이미지와 동일한 높이와 너비의 BW8 이미지입니다.

- 플렉시블 마스크에서 값이 0인 픽셀은 해당 소스 이미지 픽셀을 마스킹하여 인코딩된 이미지에 나타나지 않도록 만듭니다.
- 플렉시블 마스크에서 다른 모든 픽셀 값은 픽셀이 인코딩되도록 만듭니다.

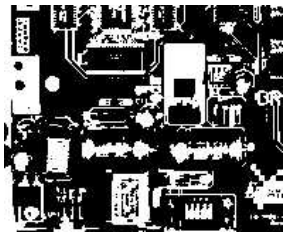
플렉시블 마스크를 만들 수 있는 EasyObject 함수



소스 이미지

1) ECodedImage2::RenderMask: 인코딩된 이미지의 레이어에서

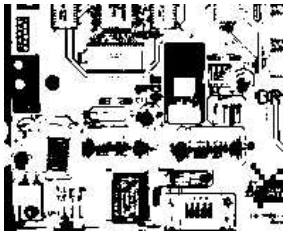
1. 플렉시블 마스크를 인코딩 및 추출하려면, 먼저 소스 이미지에서 코드화 이미지를 구축해야 합니다.
2. 분할 방식을 선택하십시오(위 이미지에는 기본 방식 `GrayscaleSingleThreshold`가 적합함).
3. 인코딩해야 하는 코드화 이미지의 레이어를 선택하십시오(예: 최소 잔여 임계값을 사용하는 흰색 및 검정색 레이어).
4. `mask.SetSize(sourceImage.GetWidth(), sourceImage.GetHeight())`를 사용하여 마스크 이미지를 원하는 크기로 만드십시오.
5. 플렉시블 마스크를 `ECodedImage2::RenderMask`에 대한 인수로 활용하십시오.



결과 BW8 이미지를 플렉시블 마스크로 사용할 수 있습니다.

2) ECodedElement::RenderMask: 블롭 또는 홀에서

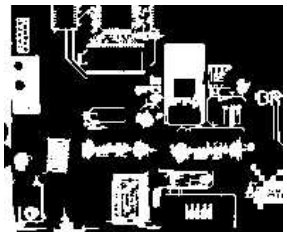
1. 관심 코드화 요소를 선택하십시오.
2. `ECodedElement::RenderMask`를 사용하여 코드화 이미지의 선택된 코드화 요소에서 마스크를 추출하여 루프를 만드십시오.
3. 또는 선택된 각 코드화 요소에 대해 특징 값을 계산할 수도 있습니다.



결과 BW8 이미지를 플렉시블 마스크로 사용할 수 있습니다.

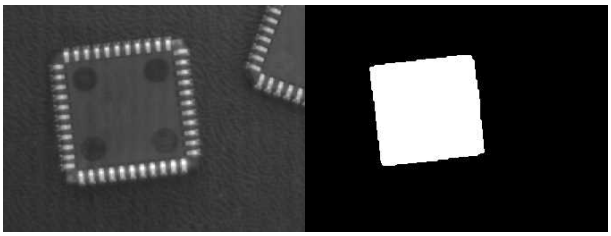
3) EObjectSelection::RenderMask: 선택적인 블롭에서

`EObjectSelection::RenderMask`는 예를 들어 노이즈로 인해 생긴 작은 개체를 폐기할 수 있습니다.



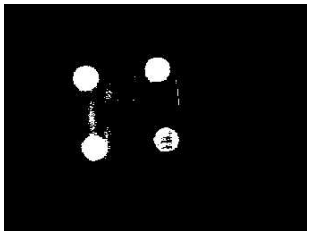
결과 BW8 이미지를 플렉시블 마스크로 사용할 수 있습니다.

예: EasyObject에 의해 인코딩되는 영역 제한



네 개의 원 찾기(왼쪽) 플렉시블 마스크로 중앙 칩을 격리할 수 있음(오른쪽)

1. 새로운 `ECodedImage2` 개체를 선언하십시오.
2. 변수 설정: 먼저 소스 이미지와 플렉시블 마스크를 선언한 다음 로드하십시오.
3. `EImageEncoder` 개체를 선언하고, 해당될 경우 적절한 세그먼터를 선택하십시오. 세그먼터를 설정하고 인코딩하기에 적절한 레이어를 선택하십시오.
4. 소스 이미지를 인코딩하십시오. 그러면 플렉시블 마스크에 포함된 영역만으로 레이어를 인코딩하는 작업이 상당히 손쉬워집니다.
   검정색 레이어에 포함된 원이 그레이스케일 단일 임계값 세그먼터에 의해 정확하게 분할된 것을 볼 수 있습니다.



5. 코드화 이미지의 모든 개체를 선택하십시오.
6. 너무 작은 개체를 필터로 제외하여 관심 개체를 선택하십시오.
7. 선택한 특징이 표시되도록 선택한 개체에 대해 반복적으로 블롭 특징을 표시하십시오.

2.11. 프로파일

프로파일 샘플링

프로파일이란 이미지 내의 선/경로/윤곽선을 따라 샘플링된 일련의 픽셀 값을 의미합니다.

- `EasyImage::ImageToLineSegment`는 지정된 선 세그먼트(임의의 방향을 향하며 이미지 내에 완전히 포함됨)를 따라 픽셀 값을 벡터로 복사합니다. 벡터 길이는 자동으로 조정됩니다. 이 함수는 플렉시블 마스크를 지원합니다.
- 경로는 벡터에 저장된 일련의 픽셀 좌표입니다.
  `EasyImage::ImageToPath`는 해당하는 픽셀 값을 벡터로 복사합니다. 이 함수는 플렉시블 마스크를 지원합니다.
- **윤곽선**은 개체의 경계를 이루는 폐쇄된 또는 폐쇄되지 않은(연결된) 경로입니다.
  `EasyImage::Contour`는 객체의 윤곽을 따라 가며 구성 픽셀 값을 프로파일 벡터 안에 저장합니다.

프로파일 분석

프로파일을 처리하여 피크 또는 변이를 찾을 수 있습니다.

- 변이는 개체 에지(검정색에서 흰색으로 또는 흰색에서 검정색으로)에 해당합니다. 변이는 신호의 첫 번째 **도함수**(변이(에지)를 피크로 변환함)를 구하고 그 안에서 피크를 찾음으로써 감지할 수 있습니다.
  `EasyImage::ProfileDerivative`는 회색조 이미지에서 추출한 프로파일의 첫 번째 도함수를 계산합니다.
  `EBW8` 데이터 유형은 무부호 값만 처리하므로, 도함수가 128을 기준으로 결정됩니다. 128 아래[위]의 값은 음[양]의 도함수에 해당합니다(감소[증가] 기울기).
- 피크는 지정된 임계값 위[또는 아래]의 신호 부분으로, 신호의 최대값 또는 최소값입니다. 이는 흰색 또는 검정색 선의 교차점이나 가늘은 특징점에 해당할 수 있습니다. 피크는 다음에 의해 정의됩니다.
  - 진폭: 임계값과 최대[또는 최소] 신호 값 사이의 차이입니다.
  - 면적: 신호 곡선과 지정된 임계값의 수직선 사이의 면입니다.

EasyImage::GetProfilePeaks 는 회색조 레벨 프로파일의 최대 및 최소 피크를 감지합니다. 노이즈로 인한 허위 피크를 없애기 위해 두 가지 선택 기준이 사용됩니다. 결과는 피크 벡터 안에 저장됩니다.

이미지에 프로파일 삽입

EasyImage::LineSegmentToImage는 벡터 또는 상수의 픽셀 값을 지정된 라인 세그먼트(임의의 방향을 향하며 이미지 내에 완전히 포함됨)의 픽셀에 복사합니다.

EasyImage::PathToImage는 벡터 또는 상수의 픽셀 값을 지정된 경로의 픽셀에 복사합니다.

3. 이미지 전처리 라이브러리

3.1. EasyImage - 그레이 스케일 이미지 전처리

EasyImage 작업은 추가적인 처리를 통해 더 나은 결과를 얻을 수 있도록 다음과 같이 이미지를 준비하는 절차입니다.

- 임계값 적용 또는 강도 변환을 통해 결함 격리
- 원근 효과 보정(병 레이블과 같이 평평하지 않은 표면의 경우)
- 플렉시블 마스크를 사용하여 복잡한 또는 단절된 형상 처리

주요 기능은 다음과 같습니다.

- Intensity Transformations(강도 변환)는 개체가 명확해지도록 각 픽셀의 회색조를 변경하는 기능입니다(히스토그램 확장).
- Thresholding(임계값 적용)은 픽셀 값을 분류하여 바이너리 이미지를 이단계 또는 삼단계 그레이스케일 이미지로 변환하는 기능입니다.
- Arithmetic and logic(산술 및 논리) 기능은 픽셀을 두 이미지로 또는 한 이미지와 하나의 상수로 조작합니다.
- Non-Linear Filtering(비선형 필터링)은 인접 픽셀의 비선형 조합을 사용하여(커널 사용) 형상을 강조하거나 노이즈를 제거하는 기능입니다.
- Geometric transforms(기하학적 변환)는 선택한 픽셀을 이동하여 재정렬, 크기 변경, 회전, 왜곡하는 기능입니다.
- Noise Reduction and Estimation(노이즈 감소 및 예측) 기능은 다른 작업(임계값 적용, 하이패스 필터링)으로 인해 노이즈가 부적절하게 강조되지 않도록 합니다.
- Gradient Scalar(그래디언트 스칼라)는 회색조 이미지에서 그래디언트 방향 또는 그래디언트 규모 맵을 생성하는 기능입니다.
- Vector operations(벡터 연산)는 이미지에서 1차원 데이터를 벡터로 추출하는 기능이며, 예를 들어 흠집이나 윤곽을 감지하거나 이미지를 명확하게 만듭니다.
- Harris corner detector(Harris 코너 감지) 기능은 BW8 이미지 내 관심 지점의 벡터를 반환합니다.
- Canny edge detector(Canny 에지 감지기)는 BW8 이미지에서 발견된 에지의 BW8 이미지를 반환합니다.
- Overlay(오버레이)는 컬러 이미지 위에 이미지를 겹칩니다.
- Operations on Interlaced Video Frames(인터페이스 비디오 프레임에 대한 작업)는 필드 재구성 또는 재정렬을 통해 인터레이스 이미지 결함을 제거합니다.
- Flexible Masks(플렉시블 마스크)는 EasyImage에서 불규칙한 형상을 처리하는 데 사용됩니다.

강도 변환

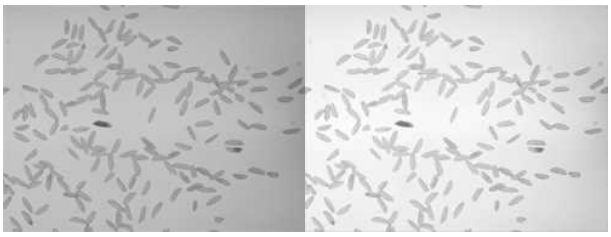
이 EasyImage 함수는 픽셀의 회색조를 변경하여 대비를 높여줍니다.

이득 오프셋

Gain Offset(이득 오프셋)은 각 픽셀을 [이전 회색 값 * 이득 오프셋 계수 + 오프셋]으로 변경합니다.

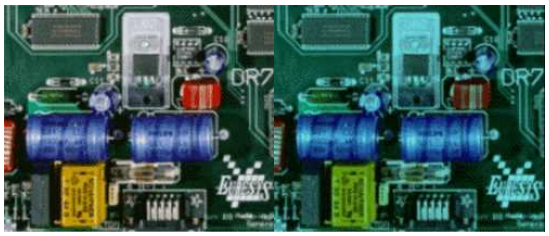
- **이득**에 따라 **대비**가 조정되며, 1에 가까운 값으로 유지되어야 합니다.
- **오프셋**은 **강도**(밝기)를 조정하며, 양수 또는 음수가 될 수 있습니다.
- 결과 값은 항상 [0..255] 범위로 포화됩니다.

이 예에서는 결과 이미지가 소스 이미지보다 대비가 좋아지고 밝아졌습니다.



소스 및 결과 이미지(이득 = 1.2 및 오프셋 = +12 적용)

컬러 이미지에는 컬러 성분(빨강, 녹색, 파랑)마다 하나씩 총 세 가지 개별적인 이득 및 오프셋 값이 있습니다.

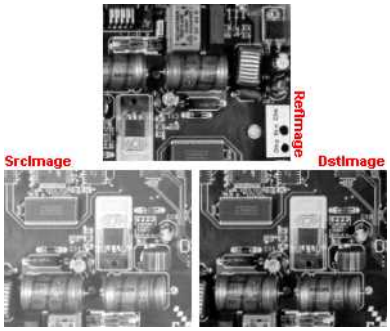


컬러 이미지에 이득/오프셋을 적용한 예

정규화

Normalize(정규화)는 동일한 구성의 이미지를 조명 조건이 다른 경우에도 비교할 수 있도록 만들어줍니다.

이 기능은 소스 이미지와 레퍼런스 이미지의 평균 회색조(밝기) 및 표준 편차(대비)를 비교합니다. 그 다음 출력 이미지가 레퍼런스 이미지와 같은 밝기와 대비가 되도록 소스 이미지에 이득과 오프셋 계수를 적용하여 정규화합니다. 이 작업에는 카메라 응답이 선형이며 이미지가 포화되지 않았다는 가정이 전제합니다.

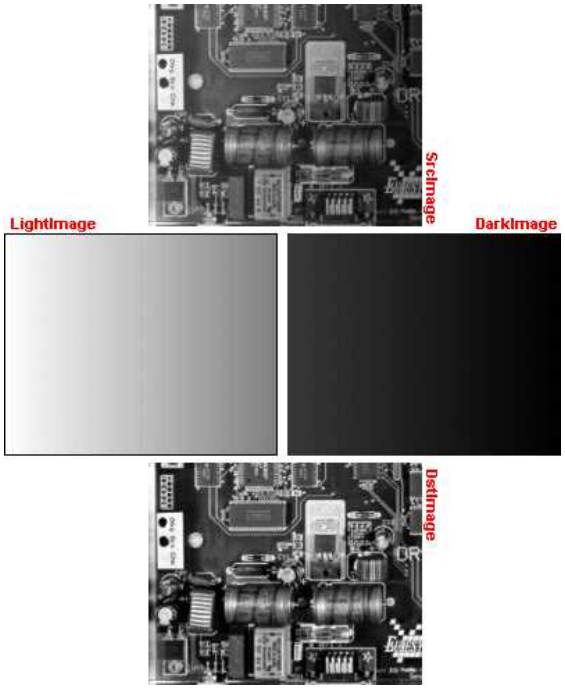


**레퍼런스 이미지(여기에서 평균 및 표준 편차가 계산됨),
소스 이미지(너무 밝음),
정규화된 이미지(대비와 밝기가 레퍼런스 이미지와 동일함)**

균일화

Uniformize(균일화)는 하나 또는 두 개의 레퍼런스 이미지로 균일하지 않은 조명 및/또는 카메라 감도를 보정하는 기능입니다. 레퍼런스 이미지에는 포화된 픽셀 값이 없어야 하며, 노이즈가 최소여야 합니다.

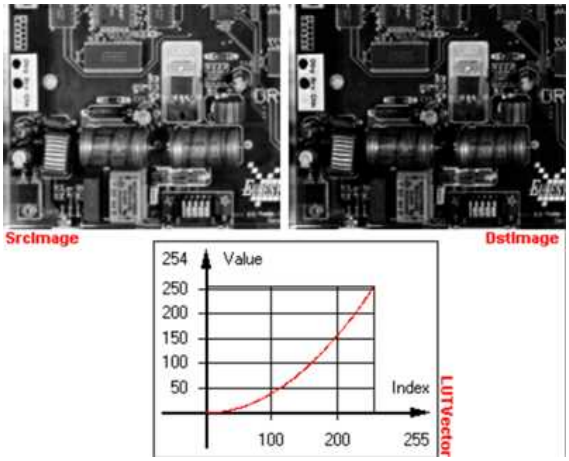
- 하나의 레퍼런스 이미지를 사용하는 경우, 어댑티브(공간-변화) 이득과 유사한 변환이 일어나며, 레퍼런스 이미지 내의 각 픽셀이 소스 이미지의 해당 픽셀에 이득을 인코딩합니다.
- 두 개의 레퍼런스 이미지를 사용하는 경우, 어댑티브 이득 및 오프셋과 유사한 변환이 일어나며, 레퍼런스 이미지 내의 각 픽셀이 소스 이미지의 해당 픽셀에 이득 또는 오프셋 중 하나를 인코딩합니다.



두 레퍼런스 이미지로 균일화된 이미지의 예

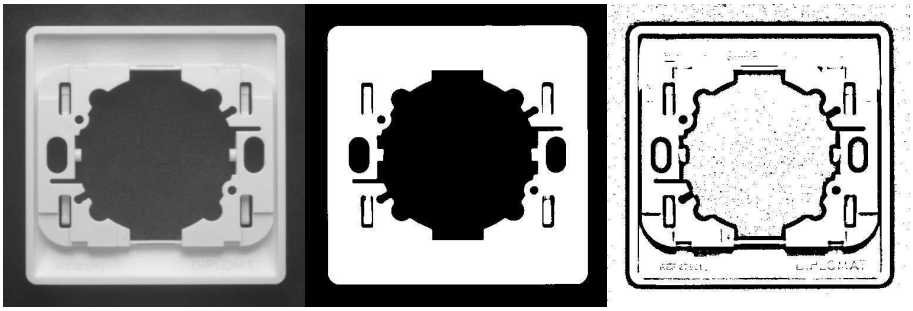
조회 테이블

Lut은 새로운 픽셀 값의 조회 테이블을 사용하여 현재 픽셀을 대체하며, BW8 및 BW16 이미지에 유용합니다. 변환 함수가 전혀 변경되지 않으면 조회 테이블을 사용하는 것이 가장 좋습니다.



변환의 예

임계값 적용



임계값 적용은 다음 메서드를 사용하여 픽셀 값을 분류함으로써 이미지를 변환하는 기능입니다.

- "자동 임계값 적용" 다음 페이지(BW8 및 BW16 이미지에 한함)
- "AutoThreshold" 페이지45(BW8 및 BW16 이미지에 한함)
- "수동 임계값 적용" 다음 페이지, 하나 또는 두 개의 임계값 사용
- "히스토그램 기반" 다음 페이지(임계값 적용 기능을 사용하기 전에 계산됨)

이 함수는 또한 임계값 아래와 위의 각 픽셀의 평균 회색조를 반환합니다.

성공적인 임계값 적용의 핵심 요소

- 개체와 배경 영역에 균일한 컬러와 조명이 적용되어야 합니다. 임계값 적용 전에 이미지 균일화 작업이 필요할 수 있습니다.

- 개체와 배경의 회색조 범위가 충분히 달라야 합니다(모든 배경 픽셀이 가장 어두운 개체 픽셀보다 어두워야 함).
- 임계값을 어떻게 지정할지 결정해야 합니다.
  - 일정: **절대** 임계값
  - 주변 조명 강도에 적응: **상대** 또는 **자동** 임계값

자동 임계값 적용

`EasyImage::Threshold` 함수와 함께 다음 중 하나의 인수를 사용할 경우 임계값이 자동으로 계산됩니다.

Min Residue(최소 잔류): 소스와 결과 이미지 사이의 이차(quadratic) 차이를 최소화합니다(임계값 함수가 인수 없이 호출된 경우 기본값).

Max Entropy(최대 엔트로피): 결과 이미지의 개체와 배경 사이의 엔트로피(예: 정보의 양)를 극대화합니다.

Isodata: 회색조의 평균이 되는 임계값, 임계값 아래 픽셀의 평균 회색조 사이 중간, 임계값 위 픽셀의 평균 회색조를 계산합니다.

수동 임계값 적용

수동 임계값 적용을 사용하려면 사용자가 하나 또는 두 개의 임계값을 입력해야 합니다.

- 소스 이미지 픽셀(BW8/BW16/C24)을 두 클래스로 분류하고 2레벨 이미지를 만들기 위한 `Threshold(임계값)` 함수에 대한 **하나의** 값. 이는 다음 중 하나가 될 수 있습니다.
  - `relativeThreshold`는 임계값 아래 픽셀의 백분율입니다. 그 다음 임계값 함수에서 적절한 임계값을 계산하거나, 또는
  - `absoluteThreshold`. 이 값은 소스 이미지 내 픽셀 값의 범위 내에 있어야 합니다.
- 소스 이미지 픽셀(BW8/BW16)을 두 클래스로 분류하고 3레벨 이미지를 만들기 위한 `DoubleThreshold` 함수에 대한 **두 개의** 값.
  - `LowThreshold`는 임계값의 하한입니다
  - `HighThreshold`는 임계값의 상한입니다

히스토그램 기반

소스 이미지의 히스토그램을 사용할 수 있는 경우, 히스토그램에서 임계값을 계산하고 (`HistogramThreshold` 또는 `HistogramThresholdBW16` 사용) 이를 수동 임계값 적용 작업에 사용하여 자동 임계값 적용 작업의 속도를 단축할 수 있습니다.

이 함수는 또한 임계값 아래와 위의 각 픽셀의 평균 회색조를 반환합니다.

AutoThreshold

원본 이미지 히스토그램을 사용할 수 없으면 `AutoThreshold`는 다음 임계값 모드: EThresholdMode_Relative, _MinResidue, _MaxEntropy and _Isodata를 사용하여 임계값을 계산할 수 있습니다..

이 함수는 플렉시블 마스크를 지원합니다.

산술 및 논리

산술 및 논리를 사용하는 이유는 다음과 같습니다.

- 픽셀 감산을 통해 이미지 사이의 **차이점 강조**(일치 확인).
- 대상 이미지를 배경만의 이미지로 나눔으로써 **불균일한 조명 보상**.
- 적절한 마스크를 준비하고, 픽셀의 논리 조합을 사용하여 해당 마스크에 속한 모든 픽셀을 지움으로써 **이미지에서 불필요한 영역 제거**.
- 두 소스 이미지의 픽셀을 합쳐 결과 이미지를 만듦으로써 **결합된 이미지 생성**.

산술 연산은 `Oper` 함수에 의해 처리되며, `EArithmeticLogicOperation` enum에 지원되는 모든 연산자가 열거됩니다.

산술 연산은 이미지와 상수에 적용할 수 있으며, 하나 또는 두 개의 소스 인수(이미지 또는 정수 상수)와 하나의 대상 인수가 포함될 수 있습니다. 소스 피연산자가 컬러 및 회색조 이미지일 경우, 각 컬러 성분이 회색조 성분과 결합되어 컬러 이미지를 제공합니다. 히스토그램 균등화를 통해 결과를 개선할 수 있습니다.

산술 및 논리 결합

허용되는 결합

	일반	복사	반전	이동	논리	오버레이	설정
상수 BW8 -> 이미지 BW8		x					
상수 C24 -> 이미지 C24		x					
이미지 BW8 -> 이미지 BW8		x	x				
이미지 BW8 -> 이미지 C24		x	x			x	
이미지 C24 -> 이미지 C24		x	x				
상수 BW8, 이미지 BW8 -> 이미지 BW8	x						
이미지 BW8, 상수 BW8 -> 이미지 BW8	x			x			x
이미지 BW8, 이미지 BW8 -> 이미지 BW8	x				x		x
이미지 BW8, 이미지 BW8 -> 이미지 C24	x					x	
상수 C24, 이미지 C24 -> 이미지 C24	x						

	일반	복사	반전	이동	논리	오버레이	설정
이미지 C24, 상수 C24 -> 이미지 C24	x			x			
이미지 C24, 이미지 C24 -> 이미지 C24	x				x		
이미지 C24, 이미지 BW8 -> 이미지 C24	x				x	x	
이미지 BW8, 이미지 C24 -> 이미지 C24	x				x		x

참고: *참고: 논리 연산자의 경우, 값이 0인 픽셀은 **FALSE(거짓)**로, 그렇지 않으면 **TRUE(참)**로 간주됩니다. **FALSE**일 경우 논리 연산의 결과는 0이며, 그렇지 않으면 255입니다.*

위 표에 있는 연산은 다음과 같이 분류됩니다.

일반

- 비교(차이의 절대값)
- 포화 합
- 포화 차이
- 포화 곱
- 포화 몫
- 모듈로
- 오버플로 프리 합
- 오버플로 프리 차이
- 오버플로 프리 곱
- 오버플로 프리 몫
- 비트 AND
- 비트 OR
- 비트 XOR
- 최소
- 최대
- 같음
- 같지 않음
- 크거나 같음
- 작거나 같음
- 더 큼
- 더 작음

복사

- 완전 복사

반전

- 반전(음)

이동

- 왼쪽 이동
- 오른쪽 이동

논리

- 논리 AND
- 논리 OR
- 논리 XOR

오버레이

- 오버레이 추가

설정

Copy if mask = 0 및 Copy if mask <> 0 연산자는 마스크 작업을 수행하는 것이 매우 편리합니다. 첫 번째 이미지 인수는 대상 이미지의 픽셀 값의 변경을 허용 또는 금지하는 마스크의 역할을 합니다.

- Copy if mask = 0
- Copy if mask <> 0

비선형 필터링

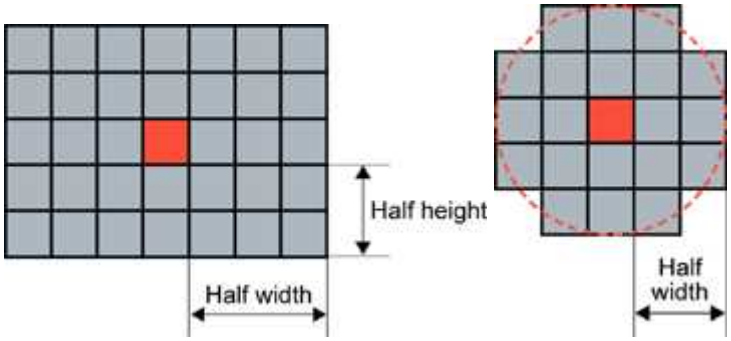
이 함수는 인접 픽셀의 비선형 조합을 사용하여 형상을 강조하거나 노이즈를 제거하는 기능입니다.

대부분은 파괴적(Top-hat 및 중앙값 필터 제외) 즉, 소스 이미지가 대상 이미지로 덮어쓰기됩니다. 파괴적 연산은 더 빠릅니다.

모두 예를 들어 `ErodeBox` 및 `BiLevelErodeBox`와 같은 회색 이미지와 2레벨 등가물을 가집니다.

1. 필요한 형상은 "커널" 다음 페이지(일반적으로 3x3 행렬에서)에 의해 정의됩니다.
2. 이 커널을 이미지 위에 밀어 넣음으로써 일치가 발견되었을 때 대상 픽셀의 값을 결정합니다.
   - 침식, 팽창: 이미지 영역을 축소/확장합니다.
   - 개방, 폐쇄: 이미지 영역 경계 픽셀을 제거/채웁니다.
   - 세선화, 농밀화: 이미지 패턴 일치를 사용하여 침식/팽창을 적용합니다.
   - Top-Hat 필터: 미세한 이미지 세부 정보를 모두 유지하면서 나머지는 모두 제거합니다.
   - 형태 정보 거리: 픽셀을 검정색으로 만들려면 얼마나 많은 침식이 필요한지 나타냅니다.
   - 형태 정보 그래디언트: 침식 및 팽창 프로세스의 외부 및 내부 에지를 나타냅니다.
   - 중앙값 필터: 임펄스 노이즈를 제거합니다.
   - Hit-and-Miss 변환: 전경/배경 픽셀의 패턴을 감지하며, 뼈대를 만들 수 있습니다.

커널



절반 너비 = 3 및 절반 높이 = 2인 직사각형 커널(왼쪽)과 절반 너비 = 2인 원형 커널(오른쪽)

형태 정보 연산자는 지정된 형상(정사각형, 직사각형, 원형)의 이웃에 포함된 픽셀 값을 결합하고, 이웃의 중심 픽셀을 그 결과로 대체합니다.결합 함수는 비선형이며, 대부분의 경우 지정된 이웃에 있는 N 값을 고려하고 점진적으로 정렬한 다음 K번째로 큰 값을 선택하는 계급 필터입니다.
가장 흔히 사용되는 세 가지 특별 사례는 침식, 팽창 *및* ***중앙값 필터이며, 여기서 K는 1(세트의 최소값), N(최대값) 또는 N/2(중앙값)가 될 수 있습니다.***

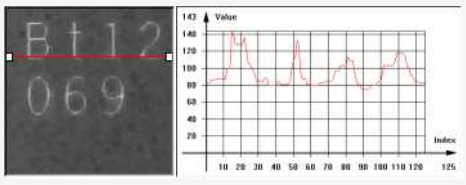
침식, 팽창, 개방 폐쇄, Top-Hat, 형태 정보 그래디언트 연산은 모두 특이 크기의 직사각형 또는 원형 커널을 사용합니다. 커널 크기는 결과에 중대한 영향을 줍니다.

예

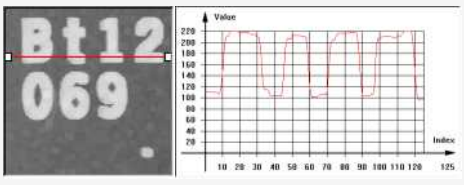
HalfWidth/HalfHeight	실제 너비/중량
0	1
1	3
2	5
3	7

침식, 팽창

침식은 흰색 개체를 감소시키고 검정색 개체를 확대하며, 팽창은 그 반대 작용을 합니다.



침식



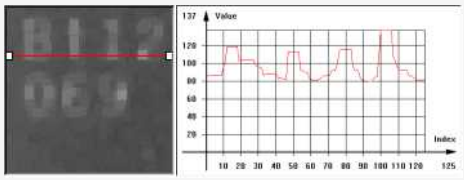
팽창

침식은 개체 에지를 따라 픽셀의 레이어를 제거함으로써 흰색 개체가 얇아지게 만듭니다. ErodeBox, ErodeDisk. 커널 크기가 증가할수록, 흰색 개체가 사라지며 검정색 개체는 두꺼워집니다.

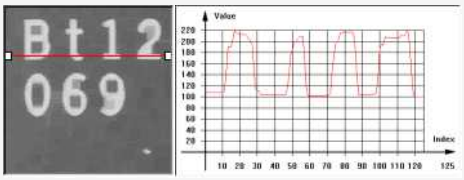
팽창은 개체 에지를 따라 픽셀의 레이어를 추가함으로써 개체가 두꺼워지게 만듭니다. `DilateBox`, `DilateDisk`. 커널 크기가 증가할수록, 흰색 개체가 두꺼워지며 검정색 개체는 사라집니다.

개방, 폐쇄

개방은 미세한 흰색 개체/티끌을 제거합니다. 폐쇄는 미세한 검정색 홀/티끌을 제거합니다.



개방



폐쇄

개방은 `OpenBox`, `OpenDisk`를 사용한 침식에 이은 팽창입니다.
전체적인 효과는 개체의 전반적인 형상을 보존하면서, 커널 크기보다 작은 흰색 세부를 제거하는 것입니다.

폐쇄는 `CloseBox`, `CloseDisk`를 사용한 팽창에 이은 침식입니다.
전체적인 효과는 개체의 전반적인 형상을 보존하면서, 커널 크기보다 작은 검정색 세부를 제거하는 것입니다.

세선화, 농밀화

이 기능은 3x3 커널을 사용하여 픽셀을 키우거나(농밀화) 또는 제거(세선화n)합니다.

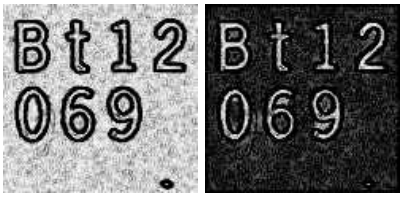
- 세선화: 라인을 단일 픽셀 두께로 줄여 에지가 감지되기 쉽도록 만듭니다.
- 농밀화: 대략적인 형상 또는 뼈대를 판단하기 쉽도록 만듭니다.

커널 계수와 픽셀 이웃 사이에 일치가 발견되면, 픽셀 값이 농밀화의 경우 255로, 세선화의 경우 0으로 설정됩니다. 커널 계수는 다음과 같습니다.

- 0: 일치하는 검정색 픽셀, 값 0
- 1: 일치하는 검정색이 아닌 픽셀, 값 > 0
- -1: 무시

Top-Hat 필터

Top-hat 필터는 불균일한 조명을 개선하는 데 탁월합니다.



흰색 Top-hat 필터: 소스 및 대상 이미지

이 필터는 이미지와 그 개방부(또는 폐쇄부) 사이의 차이를 포착합니다. 그 다음, 개방(또는 폐쇄)으로 삭제될 특징을 보존합니다. 그 결과는 커널 크기보다 작은 검정색 또는 흰색 특징만이 나타나는 완전한 단색 배경입니다.

- **흰색 Top-hat 필터**는 얇은 흰색 특징을 강조합니다. WhiteTopHatBox ,WhiteTopHatDisk.
- **검정색 Top-hat 필터**는 얇은 검정색 특징을 강조합니다.BlackTopHatBoxBlackTopHatDisk.

형태 정보 거리

Distance(거리)는 바이너리 이미지(검정색은 0, 흰색은 0 이외)의 형태 정보 거리(픽셀을 검정색으로 만드는 데 필요한 침식 경과 횟수)를 계산하고, 각 픽셀에 소스 이미지에 있는 해당 픽셀의 형태 정보 거리가 포함되는 대상 이미지를 만듭니다.

형태 정보 그래디언트

형태 정보 그래디언트는 에지 감지를 실행하며, 에지를 제외한 모든 요소를 이미지에서 제거합니다.

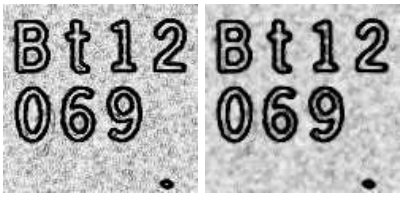
형태 정보 그래디언트는 동일한 구조적 요소를 사용하는 이미지의 팽창과 침식 사이의 차이입니다.

MorphoGradientBox, MorphoGradientDisk.

팽창 - 침식 = 그래디언트

중앙값

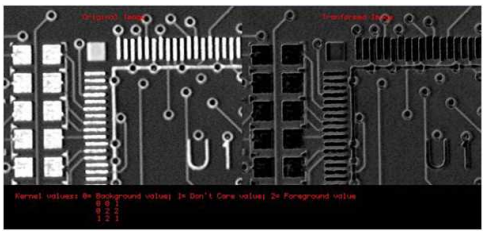
중앙 필터는 에지와 이미지 선명도를 보존하면서 임펄스 노이즈를 제거합니다.
이 필터는 3x3 정방형 커널 내에서 모든 픽셀을 이웃의 중앙값으로 대체함으로써 바깥쪽 픽셀을 폐기합니다.



중앙값 필터: 소스 및 대상 이미지

Hit-and-Miss 변환

Hit-and-miss 변환은 BW8, BW16 또는 C24 이미지 또는 ROI에서 작동하며, 이미지에 포함된 전경과 배경 픽셀의 특정 패턴을 감지합니다.



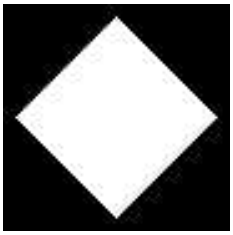
Hit-and-Miss 변환

`HitAndMiss` 함수에는 세 가지 인수가 있습니다.

- 소스 이미지 유형 `EROIBW8`, `EROIBW16` 또는 `EROIC24`에 대한 포인터
- 소스 이미지 유형에 대응하는 대상 이미지 유형에 대한 포인터
  소스와 대상 이미지의 크기는 동일해야 합니다.
- 커널 유형 `EHitAndMissKernel` 커널 개체에 대해 다음 두 가지 생성자를 사용할 수 있습니다.
  - `EHitAndMissKernel(int startX, int startY, int endX, int endY)` 여기서:
    `startX`, `startY` 는 커널의 왼쪽 상단 좌표이며 0보다 작거나 같아야 합니다.
    `endX`, `endY` 는 커널의 오른쪽 하단 좌표이며 0보다 크거나 같아야 합니다.
    생성된 커널은 그 크기와 다음과 같은 특성에 대한 명시적인 제한이 없습니다:
    `커널 너비 = (endX - startX + 1)`, `커널 높이 = (endY - startY + 1)`
  - `EHitAndMissKernel(정의 안된 int halfSizeX, unsigned int halfSizeY)` 여기서:
    `halfSizeX` 은 커널 너비의 절반의 - 1이며 반드시 0보다 커야합니다.
    `halfSizeY` 는 커널 높이의 절반의 - 1이며, 반드시 0보다 커야합니다.
    생성된 커널은 다음과 같은 특징을 갖고 있습니다:
    `커널 너비 = ((2 x halfSizeX) + 1)`, `커널 높이 = ((2 x halfSizeY) + 1)`
    `kernel StartX = - halfSizeX`, `kernel StartY = - halfSizeY`

예: 바이너리 이미지에서 코너 감지

Hit-and-miss 변환은 코너를 찾는 데 사용할 수 있습니다.

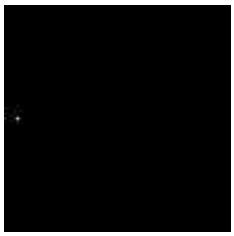


바이너리 소스 이미지

1. 왼쪽 코너를 감지하여 커널을 정의합니다. 왼쪽 코너 픽셀은 바로 왼쪽, 위, 아래에 검정색 픽셀이 있으며, 오른쪽에 흰색 픽셀이 있습니다. 다음은 왼쪽 코너를 감지하는 Hit-and-miss 커널입니다,

```
  - +
- + +
  - +
```

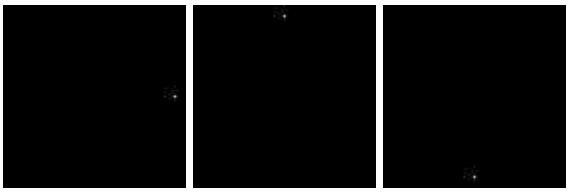
2.소스 이미지에 필터를 적용합니다. 결과 이미지가 올바른 크기여야 함에 주목하십시오.



결과 이미지, 강조된 픽셀이 마름모꼴의 왼쪽 코너에 있음

3. 나머지 세 코너도 동일한 방법으로 찾습니다. 위 필터를 회전시키는 세 커널을 찾고 적용합니다.

4.오른쪽, 위, 아래 코너를 감지합니다.



기하학적 변환

기하학적 변환은 이미지 내에서 선택한 픽셀을 이동하며, 이미지 내의 형상이 너무 크거나, 작거나, 왜곡되었거나 할 때 점대점 비교를 가능하게 해주는 유용한 기능입니다.

선택 영역은 어떤 형상이든 관계없지만 결과 이미지는 항상 직사각형입니다. 대상 이미지 내의 픽셀 중 선택 영역 밖에 대응하는 픽셀이 있는 경우는 관련성이 없는 것으로 간주되며 검정색으로 남습니다.

대상 픽셀의 소스 좌표가 정수가 아닐 경우, 보간 기법이 필요합니다.
최근접 이웃(nearest neighborhood) 방식이 가장 빠르며, 가장 가까운 소스 픽셀을 이용합니다.
이중 선형 보간(bi-linear interpolation) 방식은 더 정밀하지만 느리며, 인접한 소스 픽셀 네 개의 가중 평균을 이용합니다.

사용 가능한 기하학적 변환은 다음과 같습니다.

재정렬

잘못 정렬된 두 이미지를 재정렬하는 가장 간단한 방법은 두 이미지에 있는 특징의 위치를

정확하게 찾고(기준점 또는 피벗, 패턴 일치, 점 측정 또는 기타 사용), 특징이 서로 겹치도록 이미지 중 하나를 재정렬하는 것입니다.

한 개, 두 개 또는 세 개의 피벗 포인트를 참조 위치에 제조정하여 이미지를 등록할 수 있습니다. 최고의 정밀도가 필요하다면, 피벗 지점을 가능한 한 멀리 떨어뜨려야 합니다.

- **한 피벗 지점** 변환은 단일 평행 이동입니다. 보간 비트를 사용하는 경우 서브 픽셀 평행 이동이 가능합니다.
- **두 피벗 지점**은 평행 이동, 회전 또는 선택적인 축척 조정의 조합을 사용합니다. 축척 조정이 허용되지 않을 경우, 두 번째 피벗이 정확하게 일치하지 않는 것일 수 있습니다. 렌즈 배율 또는 시야 거리의 변경에 대응하는 경우가 아니라면, 일반적으로 축척 조정을 사용하지 않아야 합니다.
- **세 피벗 지점**은 평행 이동, 회전, 쉬어(shear) 보정 또는 선택적인 축척 조정의 조합을 사용합니다. 잘못 정렬된 라인 스캔 카메라로 이미지를 캡처하는 경우 쉬어 현상이 발생할 수 있습니다.

미러링

이 파괴적인 기능은 소스 이미지를 변경하여 다음과 같은 미러 이미지를 만듭니다.

- 수평적으로 (열이 서로 바뀜) 또는
- 수직으로(행이 서로 바뀜)

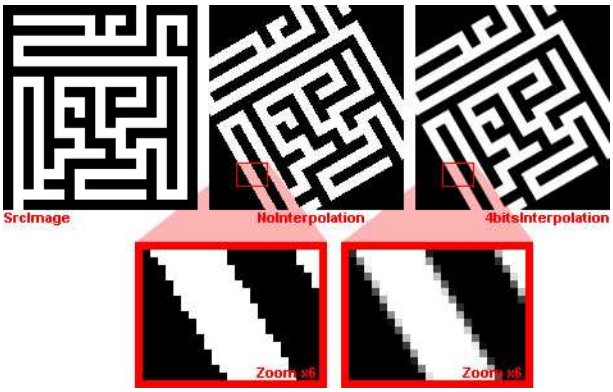
평행 이동, 축척 조정, 회전

관심 개체의 위치 또는 크기가 변경된 경우, 위치 또는 크기 변화량을 측정하고 ScaleRotate 및 Shrink 함수를 사용하여 정정된 이미지를 만들 수 있습니다.

`EasyImage::ScaleRotate`의 기능:

- 이미지 평행 이동: 소스 이미지에 있는 피벗 지점과 대상 이미지의 대응 피벗 지점의 위치 좌표를 입력합니다.
- 이미지 축척 조정: X 축과 Y 축의 축척 비율 값을 입력합니다.
- 이미지 회전: 회전 각도 값을 입력합니다.

리샘플링의 경우, 4비트 또는 8비트 정밀도의 최근접 이웃 규칙 또는 이중 선형 보간이 사용됩니다. 대상 이미지의 크기는 임의입니다.



축척 조정 및 회전 예

Shrink(축소)

`EasyImage::Shrink`: 에일리어싱 방지를 위해 사전 필터링을 적용하면서 이미지 크기를 더 작게 변경합니다.

LUT 기반 펴기

관심 특징이 형상(애너모포시스, 왜상)으로 인해 왜곡된 경우, 원형 링-쐐기 형상(예: CD 레이블의 텍스트)을 곧은 직사각형으로 펼 수 있습니다. 링-쐐기는 두 동심원과 중심을 통과하는 두 직선에 의해 한정됩니다.

`EasyImage::SetCircleWarp`는 각 픽셀을 조회 테이블로 사용되는 "펴기" 이미지에 지정된 위치로 이동하는 `EasyImage::Warp` 함수와 함께 사용할 왜곡 이미지를 준비합니다.

노이즈 감소 및 예측

노이즈는 이미지의 시각적 품질을 저하시킬 수 있으며, 특정 처리 작업(임계값 적용, 하이 패스 필터링)은 노이즈를 허용 불가능한 수준으로 강조합니다. 캡처된 이미지에는 항상 노이즈가 있습니다(픽셀 값이 실제 강도 주변으로 변동되는 라이브 이미지에서 가장 잘 관찰할 수 있음). 8비트의 정밀도로 캡처할 때, 노이즈 레벨은 일반적으로 약 3 ~ 5 회색조 값에 달합니다. 노이즈는 다음과 같은 몇 가지 형태로 구분할 수 있습니다.

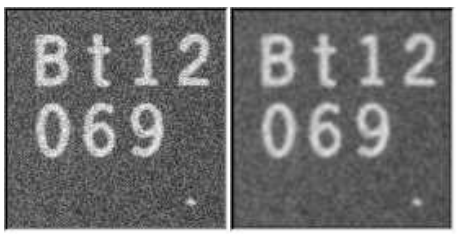
- **부가적**: 노이즈 진폭이 이미지 내용과 비례하지 않음
- **곱셈적**: 노이즈 진폭이 국소적 강도에 비례함
- **균일**: 노이즈 진폭이 0을 중심으로 한 고른 분포를 따름
- **임펄스**: 노이즈 진폭이 무한대임

임펄스 노이즈는 "점잡음(salt and pepper)" 효과를 일으키며, 균일한 노이즈는 잘 섞입니다.

공간 노이즈 감소(이미지가 1개뿐일 경우):

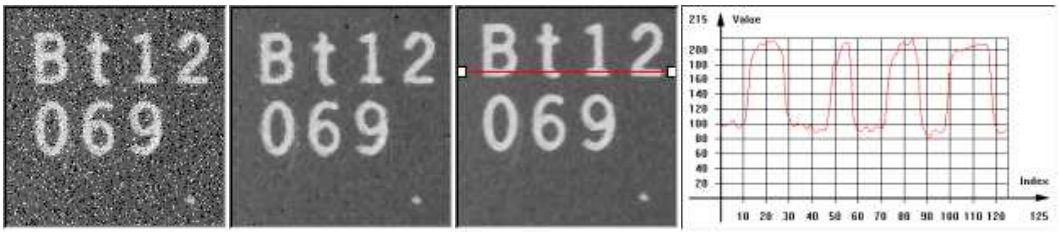
균일 및 임펄스 노이즈를 줄여주지만 에지가 흐려집니다.
노이즈와 실제 신호 변화를 구분할 수 없으므로, 항상 신호의 일부를 망칩니다.
인접 픽셀 값 사이의 상관관계를 사용하여 컨볼루션 또는 중앙값 필터링을 수행합니다.

- **컨볼루션**은 각 픽셀의 값을 이웃의 조합으로 대체함으로써 국소적 평균화를 달성합니다. 균일 노이즈를 줄이려면 선형 필터링이 권장됩니다. 이를 적용하면 에지가 흐려지는 경향이 있음에 유의하십시오.



로우 패스 필터링을 통한 균일 노이즈 감소

- **중앙값 필터링**은 각 픽셀을 픽셀 이웃(3x3 이웃에서 5번째로 큰 값)의 중앙값으로 대체합니다. 이는 임펄스 노이즈를 줄여주며 선명도는 유지합니다.



중앙값 필터링을 통한 임펄스 노이즈 감소

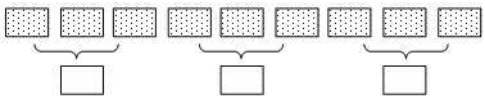
- `EasyImage::Median`
- `EasyImage::BiLevelMedian`

시간 노이즈 감소(여러 이미지, 예를 들어 이동하는 물체에 적합)

시간 노이즈 감소는 시간에 걸쳐 개별 픽셀의 연속적인 값을 결합함으로써 달성됩니다. EasyImage는 재귀 평균법과 이동 평균법을 구현합니다.

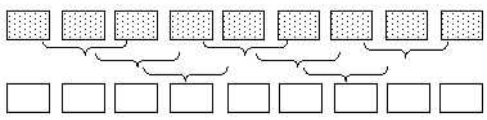
EasyImage는 몇 개의 이미지를 사용하여 노이즈를 최소화하는 방법을 세 가지 제공합니다.

- **시간 평균**: 아래 그림에 나온 것처럼 표준 산술 연산을 사용하여 단순히 N개의 이미지를 축적하고 그 평균을 냅니다. N회의 캡처 이후 평균값을 사용하여 노이즈가 제거된 이미지를 생성합니다. 노이즈는 프레임마다 변화하지만 신호는 변화 없이 유지되므로, 같은(정지) 장면의 이미지 여러 개가 있다면 노이즈를 신호에서 분리할 수 있습니다.
  N회의 캡처 후에 하나의 노이즈 제거 이미지를 생성하는 데 따른 유일한 단점은 빠른 디스플레이 갱신이 불가능하다는 것입니다.



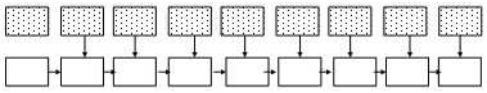
단순 평균

- **시간 이동 평균**: 최근 N개 이미지를 축적하고 새로운 이미지가 캡처될 때마다 연산 시간이 N에 의존하지 않는 방식으로 노이즈 제거 이미지를 업데이트합니다. 전체 프로세스는 `EMovingAverage`에 의해 처리됩니다. 이 방식의 단점은 노이즈가 있는 이미지가 서로 합쳐진다는 것입니다.



이동 평균

- **시간 재귀 평균**: `EasyImage::RecursiveAverage`를 사용하여 노이즈가 있는 이미지와 이전에 노이즈가 제거된 이미지를 결합합니다.



재귀 평균

재귀 평균법

이는 시간적 통합에 의한 노이즈 감소의 잘 알려진 프로세스입니다. 원리는 선형 결합을 사용하여 노이즈가 없는 이미지를 노이즈가 있는 원시 라이브 이미지 스트림과 혼합함으로써 지속적으로 업데이트하는 것입니다. 알고리즘으로 보면 다음과 같이 표현됩니다.

$$DST_N = a*Src + (1-a)*Dst_{N-1}$$

여기서 a는 혼합 계수입니다. 이 계수의 값은 지정된 노이즈 감소 비율이 달성되도록 조정할 수 있습니다.

이 절차는 정지 이미지에 적용할 때는 효과적이지만, 이동 이미지에서는 궤적 효과를 일으킵니다. 노이즈 감소 비율이 클수록 궤적 효과도 심해집니다. 이를 해결하려면 혼합 프로세스에 비선형을 도입할 수 있습니다. 연속적인 이미지 사이의 작은 회색조 값 변동은 일반적으로 노이즈이지만, 큰 변동은 이미지 변경에 해당합니다.

`EasyImage::RecursiveAverage`는 이러한 관찰을 사용하고 작은 변동에 대해 더 강력한 노이즈 제거, 그리고 그 반대를 적용합니다. 이를 통해 정지 영역에서 노이즈를, 이동 영역에서 궤적을 제거합니다.

최적의 성능을 달성하려면, `EasyImage::SetRecursiveAverageLUT` 함수를 사용하여 비선형을 한 번 사전 계산해야 합니다.

> **참고:** *`EasyImage::RecursiveAverage` 메서드를 최초로 호출하기 전에, 16비트 작업 이미지를 지워야 합니다(모든 픽셀 값을 0으로 설정).*

노이즈 예측(레퍼런스 이미지와 비교한 이미지의):

노이즈의 양을 예측하려면 둘 이상의 연속적인 이미지가 필요합니다. 가장 단순한 모드에서는 노이즈가 있는 이미지 두 개가 비교됩니다. (다른 모드도 사용 가능함: 노이즈가 없는 이미지가 있다면, 노이즈가 있는 이미지와 비교되며, 노이즈 없는 이미지는 시간 평균을 통해서도 만들 수 있음). 제곱평균제곱근 진폭 및 신호 대 잡음비를 계산합니다.

- `EasyImage::RmsNoise`는 지정된 이미지를 레퍼런스 이미지와 비교함으로써 노이즈의 제곱평균제곱근 진폭을 계산합니다. 이 함수는 플렉시블 마스크와 입력 마스크 인수를 지원합니다. 또한 BW8, BW16, C24 소스 이미지를 지원합니다.
  레퍼런스 이미지는 노이즈가 없거나(노이즈 소스를 억제하여 확보), 지정된 이미지와 동일한 분포의 노이즈에 영향을 받는 것일 수 있습니다.

- EasyImage::SignalNoiseRatio는 지정된 이미지를 레퍼런스 이미지와 비교하여 dB단위의 신호대 잡음비를 계산합니다. 이 함수는 플렉시블 마스크와 입력 마스크 인수를 지원합니다. 또한 BW8, BW16, C24 소스 이미지를 지원합니다.
  레퍼런스 이미지는 노이즈가 없거나(노이즈 소스를 억제하여 확보), 지정된 이미지와 동일한 분포의 노이즈에 영향을 받는 것일 수 있습니다.

신호 진폭은 제곱 픽셀 회색조 값의 합계입니다.

노이즈 진폭은 지정된 이미지와 레퍼런스 이미지의 픽셀 회색조 값 사이의 제곱 차의 합계입니다.

스칼라 그래디언트

EasyImage::GradientScalar는 지정된 소스 이미지에서 파생된 (스칼라) 그래디언트 이미지를 계산합니다.

그래디언트에서 파생되는 스칼라 값은 사전 설정 조회 테이블 이미지에 따라 결정됩니다.

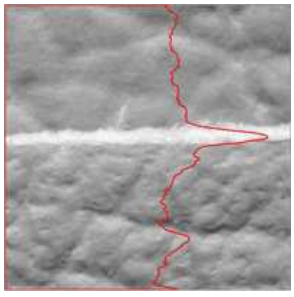
그레이스케일 이미지의 그래디언트는 벡터에 해당하며, 그 성분은 수평 및 수직 방향의 회색조 신호의 부분 도함수입니다. 벡터는 방향 및 길이, 그래디언트 방향에 대한 일치, 그래디언트 규모로 특성화할 수 있습니다.

이 기능은 회색조 이미지에서 그래디언트 방향 또는 그래디언트 규모 맵(회색조 이미지)을 생성합니다.
효율성을 위해, 원하는 변환을 정의하는 데에 사전 계산된 조회 테이블이 사용됩니다.
이 조회 테이블은 표준 EImageBW8/EImageBW16 형태로 저장됩니다.
EasyImage::ArgumentImage 또는 EasyImage::ModulusImage를 GradientScalar를 호출하기 전에 한 번 사용하십시오.

벡터 작업

이미지에서 1차원 데이터를 추출하면 벡터로 처리되는 데이터의 선형 집합이 생성됩니다. 데이터 양이 감소되므로 이후의 작업은 빠르게 처리됩니다. 메서드는 다음 중 하나입니다.

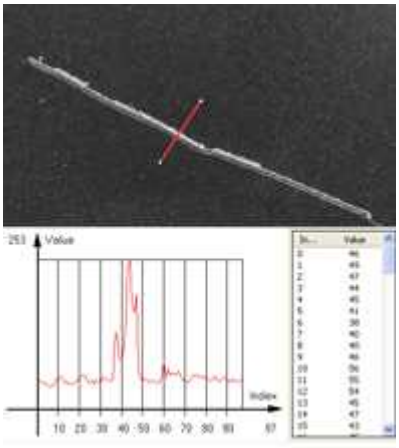
프로젝션



모든 회색 컬러 레벨 값의 합계 또는 평균을 지정된 방향에 다양한 벡터 유형으로 투영합니다(EBW32Vector로 투영하는 경우 레벨이 추가되며, EBW8Vector, EBW16Vector 또는 EC24Vector로 투영하는 경우 평준화됨). 이 함수는 **플렉시블 마스크**를 지원합니다.

- EasyImage::ProjectOnAColumn은 이미지를 수평 방향으로 열에 투영합니다.
- EasyImage::ProjectOnARow는 이미지를 수직 방향으로 행에 투영합니다.

프로파일



지정된 세그먼트, 경로 또는 윤곽선을 따라 일련의 픽셀 값을 샘플링하고, 그 피크와 변이를 분석 및 수정하여 이미지를 더 명확하게 만듭니다.

1. 라인 세그먼트/경로/윤곽선의 프로파일을 확보하십시오.

   EasyImage::ImageToLineSegment는 지정된 라인 세그먼트(임의의 방향을 향함)를 따라 픽셀 값을 벡터로 복사합니다. 라인 세그먼트는 이미지 내에 완전히 포함되어 있어야 합니다. 벡터 길이는 자동으로 조정됩니다. 이 함수는 플렉시블 마스크를 지원합니다.

   EasyImage::ImageToPath 는 해당 픽셀 값을 벡터에 복사합니다. 이 함수는 플렉시블 마스크를 지원합니다. 경로 는 벡터에 저장된 일련의 픽셀 좌표입니다.

   EasyImage::Contour는 개체의 윤곽선을 따라가면서 그 구성 픽셀 값을 프로파일 벡터 내에 저장합니다. **윤곽선**은 개체의 경계를 이루는 폐쇄된 또는 폐쇄되지 않은(연결된) 경로입니다.

2. 프로파일을 분석하여 피크 또는 변이를 찾으십시오.

   EasyImage::GetProfilePeaks 는 회색조 레벨 프로파일의 최대 및 최소 피크를 감지합니다. 노이즈로 인한 허위 피크를 없애기 위해 두 가지 선택 기준이 사용됩니다. 결과는 피크 벡터 안에 저장됩니다.
   피크는 흰색 또는 검정색 선 또는 얇은 특징의 교차점에 해당할 수 있는 신호의 최대 또는 최소값입니다. 피크는 다음에 의해 정의됩니다.
   - 진폭: 임계 값과 최대 [또는 최소] 신호 값 간의 차이.
   - 면적: 주어진 임계 값에서 신호 곡선과 수평선 사이의 표면.

   변이는 개체 에지(검정색에서 흰색으로 또는 흰색에서 검정색으로)에 해당합니다. 변이는 신호의 첫 번째 **도함수**를 구하고 그 안에서 피크를 찾음으로써 감지할 수 있습니다.
   EasyImage::ProfileDerivative는 회색조 이미지에서 추출한 프로파일의 첫 번째 도함수를 계산합니다. 이 도함수는 변이(에지)를 피크로 변환합니다.
   EBW8 데이터 유형은 무부호 값만 처리하므로, 도함수가 128을 기준으로 결정됩니다. 128 아래의 값은 음의 도함수(감소 기울기)에 해당하며, 128 위의 값은 양의 도함수(증가 기울기)에 해당합니다.

3. 프로파일을 이미지에 삽입하십시오.

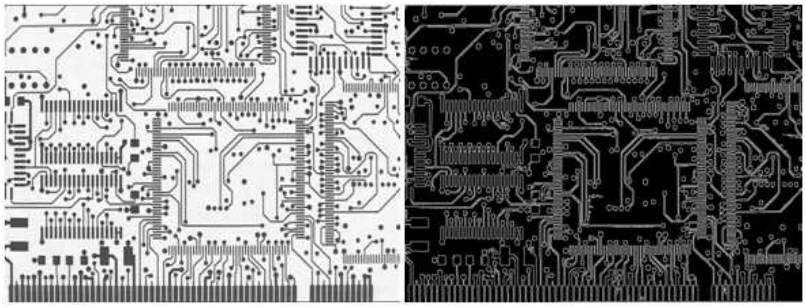
   `EasyImage::LineSegmentToImage`는 벡터 또는 상수의 픽셀 값을 지정된 라인 세그먼트(임의의 방향을 향함)의 픽셀에 복사합니다. 라인 세그먼트는 이미지 내에 완전히 포함되어 있어야 합니다.

   `EasyImage::PathToImage`는 벡터 또는 상수의 픽셀 값을 지정된 경로의 픽셀에 복사합니다.

Canny 에지 감지기

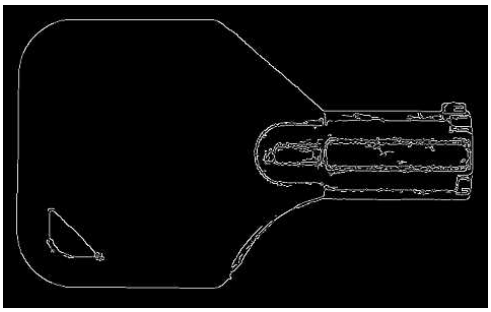
Canny 에지 감지기는 다음과 같은 기능을 지원합니다.

- 뛰어난 감지 기능: 모든 에지 발견 가능
- 뛰어난 국소화: 발견된 에지가 이미지 내의 "실제" 에지와 최대한 가까운 위치입니다.
- 최소 응답: 각 위치에 대해 하나의 에지 응답이 용인되므로, 근접하거나 상호 교차되는 에지 응답을 피할 수 있습니다



Canny 에지 감지 후의 소스 이미지 및 결과

EasyImage Canny 에지 감지기는 그레이스케일 BW8 이미지로 작업을 수행하고, 픽셀의 값이 **0**과 **255** 두 가지 뿐인 흑백 BW8 이미지를 제공합니다. 소스 이미지 내의 에지에 해당하는 픽셀은 **255**로 설정되며 다른 모든 픽셀은 **0**으로 설정됩니다. 이 기능은 축척 분석을 조정할 수 있으며, 서브 픽셀 보간은 허용하지 않고, 임계값 적용 후 바이너리 이미지를 제공합니다.



Canny 에지 감지기 예제

Canny 에지 감지기에는 다음 두 매개변수만 필요합니다.

- **관심있는 형상의 특징적 스케일**: 소스 이미지를 매끄럽게 하는 데 사용된 가우시안 필터의 표준 편차.
- **히스테리시스 있는 그래디언트 임계값**: 소스 이미지 그래디언트의 최대 크기는 0에서 1 사이의 비율(두 값)으로 표현됨.

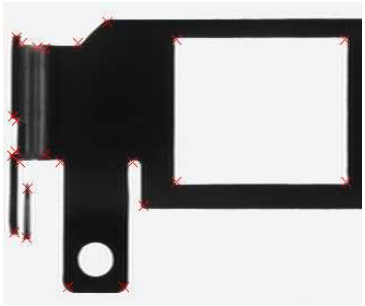
Canny 에지 감지기의 API는 단일 클래스, `ECannyEdgeDetector`이며 다음과 같은 메서드가 포함됩니다.

- `Apply`: 이미지/ROI에 Canny 에지 감지기를 적용합니다.
- `GetHighThreshold`: 에지로 간주하려는 픽셀의 상한 히스테리시스 임계값을 반환합니다.
- `GetLowThreshold`: 에지로 간주하려는 픽셀의 하한 히스테리시스 임계값을 반환합니다.
- `GetSmoothingScale`: 관심 형상의 축척을 반환합니다.
- `GetThresholdingMode`: 히스테리시스 임계값의 모드를 반환합니다.
- `ResetSmoothingScale`: Gaussian 필터를 사용한 소스 이미지의 평탄화를 방지합니다.
- `SetHighThreshold`: 에지로 간주하려는 픽셀의 상한 히스테리시스 임계값을 설정합니다.
- `SetLowThreshold`: 에지로 간주하려는 픽셀의 하한 히스테리시스 임계값을 설정합니다.
- `SetSmoothingScale`: 관심 형상의 축척을 설정합니다.
- `SetThresholdingMode`: 히스테리시스 임계값의 모드를 설정합니다.

결과 이미지의 크기는 입력 이미지와 동일해야 합니다.

Harris 코너 감지기

Harris 코너 감지기는 회전, 조명 변화, 이미지 노이즈에 따른 변화가 없으며, 그레이스케일 BW8 이미지에서 작동하며 관심 지점의 벡터를 제공합니다.



Harris 코너 감지기의 예

EasyImage Harris 코너 감지기에는 세 가지 매개변수가 필요합니다.

- 적분 척도 σi: 축척 분석에 사용되는 가우시안 필터의 표준 편차입니다.
  σd = 0,7 x σi, 여기서 σd는 미분 척도이며, 그래디언트 계산 도중 노이즈 감소에 사용되는 가우시안 필터의 표준 편차입니다.
- 코너 임계값: 소스 이미지의 코너성(cornerness) 최대값을 나타내는 0 ~ 1 사이의 분수입니다.
- 서브 픽셀 감지를 켜고 끄는 부울 함수.

각 관심 지점에서 다음 특성을 구할 수 있습니다.

- 중심 위치(활성화된 경우 서브 픽셀 정밀도의 픽셀 좌표).
- 코너성 측정치.
- 미분 척도 σd에 대한 그래디언트 규모.
- 미분 척도 σd에 대한 X 축상의 그래디언트 값.
- 미분 척도 σd에 대한 Y 축상의 그래디언트 값.

Harris 코너 감지기의 API는 EHarrisCornerDetector라는 이름의 단일 클래스와 다음 메서드로 구성됩니다.

- Apply: 이미지/ROI에 Harris 코너 감지기를 적용합니다.
- EHarrisCornerDetector: 기본값으로 초기화된 EHarrisCornerDetector 개체를 만듭니다.
- GetDerivationScale: 현재 도함수 척도를 반환합니다.
- GetScale: 적분 척도를 반환합니다.
- GetThreshold: 현재 임계값을 반환합니다.
- GetThresholdingMode: 코너성 측정에 사용되는 현재 임계값 모드를 반환합니다.
- IsGradientNormalizationEnabled: 코너성 측정치의 계산 전에 그래디언트를 정규화할 것인지 여부를 반환합니다.
- IsSubpixelPrecisionEnabled: 서브 픽셀 보간의 활성화 여부를 반환합니다.
- SetDerivationScale: 도함수 척도를 설정합니다.
- SetGradientNormalizationEnabled: 코너성 측정치의 계산 전에 그래디언트를 정규화할 것인지 설정합니다.
- SetScale: 적분 척도를 설정합니다.
- SetSubpixelPrecisionEnabled: 서브 픽셀 보간의 활성화 여부를 설정합니다.
- SetThreshold: 코너로 간주하려는 픽셀의 코너성 측정에 적용할 임계값을 설정합니다.
- SetThresholdingMode: 코너성 측정에 사용할 임계값 모드를 설정합니다.

Harris 코너 감지기의 기본적인 사용법

EHarrisCornerDetector 클래스의 개체를 Harris 감지기 애플리케이션 전반에서 재사용하여 설정 시간을 단축할 수 있습니다.

1. **감지기의 인스턴스를 만들고** 적절한 메서드를 설정합니다. 예를 들어, 적분 척도, SetScale, 2픽셀의 공간 한도를 가질 수 있는 관심 구조.
2. 새 이미지에 두 개의 인수를 갖는**감지기**를 적용합니다: 입력 이미지와 입력 이미지 EHarrisInterestPoints에 있는 관심 지점.
3. 출력 벡터의 개별 요소를 액세스합니다.

오버레이

EasyImage::Overlay는 지정된 위치에서 컬러 이미지 위에 이미지를 겹칩니다.

컬러 이미지가 소스 이미지로 제공되는 경우, 레퍼런스 컬러와 동일한 픽셀을 제외하고 해당 이미지의 모든 픽셀이 대상 이미지로 복사됩니다. C24 이미지를 오버레이 소스 이미지로 사용하는 경우, 대상 이미지 내의 오버레이 컬러가 오버레이 소스 이미지 내의 컬러와 동일하므로 다중 컬러 오버레이가 가능합니다.

BW8 이미지가 소스 이미지로 제공되는 경우, 소스 이미지로 교체되는 레퍼런스 컬러를 제외하고 모든 오버레이 이미지 픽셀이 대상 이미지로 복사됩니다.

이 함수는 플렉시블 마스크와 입력 마스크 인수를 지원합니다. 또한 C24, C15, C16 소스 이미지를 지원합니다.

인터레이스 비디오 프레임에 대한 작업

이미지가 인터레이스될 때는, 두 프레임(짝수 및 홀수 라인)이 동시에 기록되지 않습니다. 장면에 움직임이 있다면 눈에 띄는 결함이 발생할 수 있습니다(개체의 에지에 "빗살(comb)" 효과가 나타남).

`EasyImage::RealignFrame`은 동작이 균일하고 수평 방향일 때(컨베이어 벨트 위의 물체) 프레임 하나를 수평으로 이동하여 이러한 문제를 해소합니다. 이동 범위는 자동으로 예측할 수 있습니다.

`EasyImage::GetFrame`은 이미지에서 지정된 패리티의 프레임을 추출하며,
`EasyImage::SetFrame`은 이미지에 포함된 지정된 패리티의 프레임을 교체합니다.

`EasyImage::MatchFrames`는 이미지의 두 연속 라인을 비교하여 최적의 이동 범위를 결정합니다. 이 두 라인은 일부 에지 또는 불균일한 영역을 교차하도록 선택됩니다.

`EasyImage::RebuildFrame`은 다른 프레임의 라인 사이의 보간을 통해 이미지의 한 프레임을 재구성합니다.

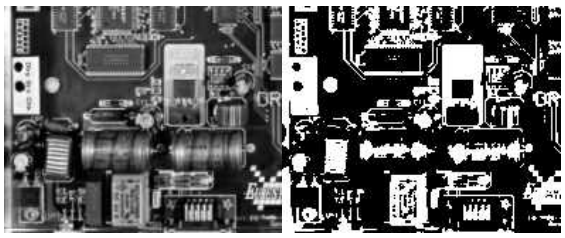
`EasyImage::SwapFrames`은 이미지의 짝수 행과 홀수 행을 서로 교환합니다. 이는 인터레이스 이미지의 캡처 과정에서 짝수와 홀수 프레임이 뒤섞였을 때 유용합니다.

이동된 행만 복사되므로 동일한 이미지를 소스와 대상으로 사용해야 합니다. 다른 대상 이미지를 사용하려면, 먼저 소스 이미지를 대상 이미지 개체에 복사해야 합니다.

대상 이미지의 크기는 다음과 같이 결정됩니다.

```
dstImage_Width = srcImage_Width
dstImage_Height = (srcImage_Height + 1 - odd ) / 2
```

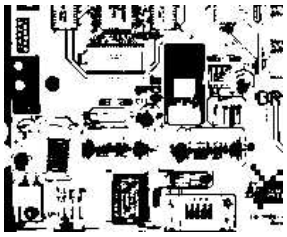
EasyImage에서 플렉시블 마스크 사용



소스 이미지(왼쪽)와 마스크 변수(오른쪽)

Easyimage에서 플렉시블 마스크를 사용하는 간단한 절차

1. **EasyImage에서 입력 마스크를 인수로 채택하는 함수를 호출하십시오**. 예를 들어, 흰색 레이어에 이어 검정색 레이어에 포함된 픽셀의 평균값을 평가할 수 있습니다.
2. **결과가 표시됩니다.**



결과 이미지

플렉시블 마스크를 지원하는 EasyImage 함수

- `EImageEncoder::Encode`에는 BW1, BW8, BW16, C24 소스 이미지에 대응하는 플렉시블 마스크 인수가 있습니다.
- AutoThreshold.
- 히스토그램 (`HistogramThreshold` 함수에는 마스크 인수를 포함한 오버로드가 없습니다).
- RmsNoise, SignalNoiseRatio.
- 오버레이(BW8 소스 이미지에는 마스크 인수를 포함한 오버로드 없음).
- ProjectOnAColumn, ProjectOnARow(벡터 프로젝션).
- ImageToLineSegment, ImageToPath(벡터 프로파일).

3.2. EasyImage - 그레이 스케일 이미지 전처리

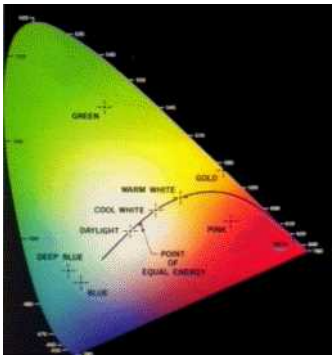
EasyColor는 객체를 감지, 분류 및 분석하여 가능한 한 효율적으로 컬러 이미지 처리를 수행합니다. 여러 가지 변환 기능을 제공하므로 모든 컬러 시스템을 처리할 수 있습니다.

컬러 정의 및 지원 시스템

컬러란 무엇일까요?

인간의 눈은 빛에 민감합니다.

- **강도** 또는 **무채색** 감각, **그레이스케일** 이미지에 의해 포착됨.
- **파장** 또는 **색채** 감각, 적색, 녹색, 청색의 기본**색상**으로 표현됨.
  트루 컬러 디지털 이미지(픽셀당 24비트, RGB 채널당 8비트)는 육안으로 구별할 수 있는 모든 컬러를 표현합니다.



XYZ 색 공간 내의 가시 색상 영역

컬러 시스템에는 다음 세 가지가 있습니다.

- **혼합** 시스템(RGB/XYZ)은 삼원색의 비율을 조합하는 시스템입니다.
- **YUV 루마/크로마** 시스템(XYZ/YUV)은 무채색(Y)과 유채색 감각(U 및 V)을 분리한 시스템입니다. 흑백 이미지도 필요한 경우에 사용합니다(텔레비전).
- **강도/채도/색조** 시스템(RGB/XYZ/YUV)은 확장 유채색(컬러 채도 및 색조) 감각에서 무채색(흑백 강도)을 분리한 시스템입니다. 조명 효과를 제거하거나, RGB 이미지를 다른 컬러 시스템으로 변환할 때 사용합니다. 채도가 높은 컬러는 더 생생하며, 채도가 낮은 컬러는 더 회색을 띱니다.

일반적인 용도:

- RGB는 모니터, 카메라 및 기타 디스플레이 장치에 사용됩니다.
- YUV는 색차 정보를 압축하여 컬러 이미지를 효율적으로 전송하는 데 사용됩니다.
- XYZ는 장치와 독립적인 컬러 표현에 사용됩니다.

모든 이미지 처리 작업에 **양자화** 좌표 즉, 이미지를 프레임 버퍼에 저장하는 데 바이트 표현을 사용하는 [0..255] 구간 내의 불연속 값을 사용할 수 있습니다.

컬러 시스템 변환 작업에도 흔히 [0..1] 구간으로 정규화되는 연속 값인 간단한 **비양자화** 좌표를 사용할 수 있습니다.

컬러 이미지 처리

컬러 이미지는 픽셀당 세 성분으로 이루어진 벡터 필드입니다. 개체에 의해 반영되는 세 RGB 성분 모두는 광원의 강도에 비례하는 크기를 갖습니다. 두 컬러 성분의 비율을 고려하면 조명과 독립적인 이미지를 얻을 수 있습니다. 픽셀당 세 부분의 정보를 현명하게 조합하면 더 뛰어난 특성을 추출할 수 있습니다.

컬러 이미지는 다음과 같은 세 가지 방식으로 처리할 수 있습니다.

- **성분 추출**: 세 가지 컬러 정보에서 가장 관련성 높은 특징을 추출하여 데이터의 양을 줄일 수 있습니다. 예를 들어, 개체가 색조에 따라 구분되는 경우, 전처리 단계에서 이미지를 색조 값만 포함된 회색조 이미지로 변환할 수 있습니다.
- **비연계 변환**: 각 컬러 성분에 대해 개별적으로 작업을 수행할 수 있습니다. 예를 들어, 두 이미지를 함께 더하면 빨강, 녹색, 파랑 성분이 합쳐지고, 결과가 성분별로 결과 컬러 이미지에 저장됩니다.

- **연계 변환**: 모든 세 컬러 성분을 결합하여 세 가지 파생 성분을 만들 수 있습니다. 예를 들어 YIQ를 RGB로 변환하는 경우에 해당합니다.

지원되는 컬러 시스템

EasyColor는 RGB, XYZ, L*a*b*, L*u*v*, YUV, YIQ, LCH, ISH/LSH, VSH, YSH 컬러 시스템을 지원합니다.

24비트 Windows 비트맵과 호환되므로 RGB가 선호되는 내부 표현 방식입니다.

	RGB 기반	XYZ 기반	YUV 기반
혼합	RGB	XYZ	—
루마/크로마	—	L*a*b* L*u*v*	YUV YIQ
강도/채도/색조	ISH LSH VSH	LCH	YSH

LUT(조회 테이블)를 사용한 변환

EasyColors 조회 테이블은 지정한 입력에 대해 어떤 출력이 대응하는지 정의하는 값의 배열을 제공하므로, 사용자 정의 변환 방식으로 이미지를 변경할 수 있습니다.

컬러 픽셀은 16,777,216(2^{24})개의 값을 가질 수 있으며, 이러한 항목이 포함된 풀 컬러 LUT는 50 MB의 메모리를 차지하게 되고 변환에 막대한 시간이 소요됩니다. 사전 계산된 LUT 덕분에 컬러 변환이 실현 가능하게 되었습니다.

컬러 이미지를 변환하려면 다음 중 한 가지 기능을 사용하여 LUT를 초기화해야 합니다.

"이득/오프셋용 LUT(컬러) " 페이지69: `EasyImage::GainOffset`,

"컬러 교정용 LUT" 페이지69: `Calibrate`,

"컬러 밸런스용 LUT" 페이지69: `WhiteBalance`,

`ConvertFromRGB`, `ConvertToRGB`.

이 색상 LUT는`EasyColor::Transform`과 같은 변환 작업에 사용되거나 정량화 되지 않은 값(연속, [0,1]로 정규하됨)을 사용하는 `EColorLookup` 을 사용하여 사용자 정의 변환을 만들고 소스 및 대상 시스탬을 지정할 수 있습니다. 일부 작업에는 변환된 이미지가 저장되는 것을 방지하기 위해 즉석 LUT가 사용되며, 예를 들면 이미지가 RGB 형식일 때 U(YUV 중) 성분을 변경하는 경우가 있습니다.

정밀도 및 속도의 최적의 조합은 `IndexBits` 및 `Interpolation`의 선택에 따라 결정되며, 변환된 값의 정밀도는 대체로 색인 비트의 수에 비례합니다.

- 테이블 항목의 수가 적을수록 저장 필요성이 감소하지만, 정밀도 또한 낮아집니다.
- 보간을 사용하지 않으면 실행 시간이 빨라지지만 정밀도가 낮아집니다. 보간을 사용하면 성분당 8비트의 정밀도를 복구할 수 있습니다. 관련된 변환이 선형(예: YUV에서 RGB로)일 경우, 테이블 항목의 수와 관계없이 보간으로 항상 정확한 결과를 얻을 수 있습니다.

색인 비트	항목 수	테이블 크기(바이트)
4	$2^{(3*4)}$ = 4,096	14,739
5	$2^{(3*5)}$ = 32,768	107,811
6	$2^{(3*6)}$ = 262,144	823,875

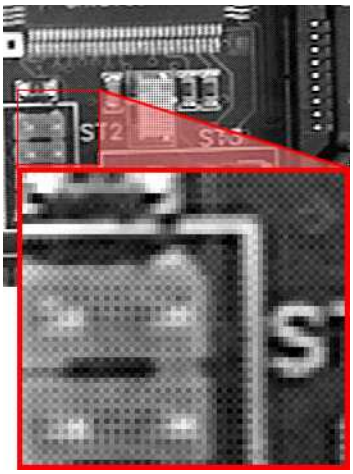
불연속 양자화 대 연속 비양자화 비교

기존 시스템의 색 좌표는 일반적으로 **연속되는** 값이며, 흔히 **[0..1]** 구간으로 정규화됩니다. 이러한 값에 대한 연산은 **비양자화**라고 부르며 간단한 편입니다.

하지만, 프레임 버퍼에 이미지를 저장하려면 **[0..255]** 구간의 **불연속** 값에 대응하는 바이트 표현이 필요합니다. 이러한 값을 **양자화**라고 부릅니다.

모든 이미지 처리 작업이 양자화 값에 적용되지만, 변환 작업은 비양자화 좌표를 사용하여 지정할 수도 있습니다.

Bayer 변환



Bayer 패턴 인코딩 이미지

Bayer 인코딩 이미지는 트루 컬러 이미지(EC24)와 호환되지 않지만,
`EasyColor::BayerToC24`에서 `EColorLookup` 매개변수를 사용하여 화이트 밸런스 및 감마 수정을 적용할 수 있습니다.
Bayer 이미지는 3배 가량 작으므로 프로세스가 훨씬 빠릅니다.

EasyObject에서 Bayer 패턴을 사용하여 컬러 이미지를 만들 수 있습니다.

YUV444 / YUV422 변환

YUV 이미지는 `Format444To422` 함수를 사용하여 4:4:4에서 4:2:2 형식으로(또는 `422` 형식에서 `444` 형식으로 변환 가능) 변환하면 시각적 품질의 저하 없이 최소화할 수 있습니다.

- 4:4:4 형식은 픽셀당 3바이트의 정보를 사용합니다.
- 4:2:2 형식은 픽셀당 2바이트의 정보를 사용합니다.
  이 형식은 다음과 같이 U 및 V 크로마의 **짝수** 픽셀을 Y 루마의 **짝수 및 홀수** 픽셀과 함께 저장합니다.

Y[짝수] U[짝수] Y[홀수] V[짝수]

병합, 추출 및 컬러

컬러 이미지에는 연속 색조 이미지의 세 컬러 평면이 포함됩니다.
회색조 이미지가 컬러 시스템의 성분이 될 수 있습니다.

성분 병합 및 추출

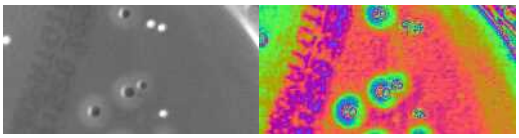
EasyColor에서 한 번에 한 평면을 또는 세 평면 모두를 함께 변경하거나 추출할 수 있습니다. `Compose`, `Decompose`, `GetComponent`, `SetComponent`를 참조하십시오.

이 작업에서 컬러 LUT를 사용하여 즉석에서 변환할 수 있으며, 명도, 채도, 색조 평면으로부터 RGB 이미지를 구축할 수 있습니다.

> **참고:** *EasyColor 기능은 Windows 비트맵 형식의 인터리브 컬러 평면(파랑, 녹색, 빨강 픽셀이 각각 서로 이어짐)을 지원하는 데 필요한 인터리브/언인터리브 작업을 수행합니다.*

유사(Pseudo) 컬러를 사용한 회색조 이미지의 컬러 변환

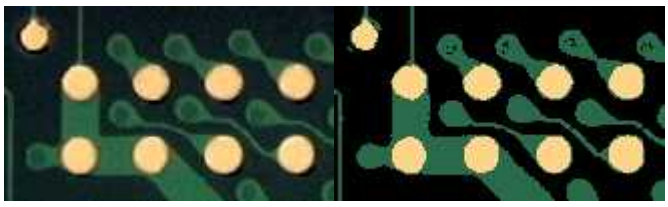
그 요령은 256 컬러의 정규 색역을 정의하여 각 컬러가 해당 회색조 값으로 픽셀에 할당되도록 하는 것입니다.
유사 컬러 음영을 정의하려면 임의 시스템의 색 공간 내에서 궤도를 지정해야 합니다. 그리고 그리기 기능 색상표(이미지 및 벡터 그리기 참조)를 사용하여 유사 컬러를 정의한 다음, 다른 컬러 이미지처럼 저장하거나 변환할 수 있습니다.



회색조 및 유사 컬러 이미지

컬러 개체 분리

이 EasyColor 프로세스는 EasyObject 에서 더 나은 블롭 생성을 위해 레이블이 적용된 이미지 세그먼터와 사용할 수 있는 레이어 색인을 사용하여 일련의 고유 컬러를 선택하고 각 픽셀을 가장 가까운 컬러와 연결합니다.



원시(raw) 이미지 및 세그먼트 이미지(3색)

Bayer 변환

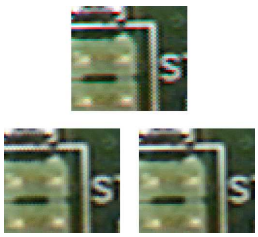
Bayer 패턴은 단일 센서에서 컬러 정보를 캡처하는 데 사용되는 컬러 이미지 인코딩 형식입니다.
일부 픽셀은 빨간 빛만, 다른 픽셀은 녹색 또는 파란 빛만 받아들이도록 센서 앞에 특정 레이아웃의 컬러 필터가 배치됩니다.
Bayer 패턴으로 인코딩된 이미지는 회색조 이미지와 동일한 형식이며, 3배 더 적은 정보를 전달합니다. 실제 수평 및 수직 해상도는 트루 컬러 이미지보다 낮습니다.

Bayer 대 트루 컬러 형식 비교

> **참고:** *Bayer 패턴은 일반적으로 왼쪽 위 코너의 GB/RG 블록으로 시작됩니다. 이미지를 자를 때 이 패리티 규칙이 상실될 수 있지만, Open eVision ROI을 사용할 때는 패리티 조정이 필요하지 않습니다.*

Bayer 변환 메서드 `EasyColor::BayerToC24`는 Bayer 패턴을 사용하여 캡처하고 회색조 이미지로 저장된 이미지를 트루 컬러 이미지로 변환합니다. 누락된 픽셀은 세 가지 방법으로 재생할 수 있습니다. 보간 작업이 복잡할수록 변환이 느려집니다. 하지만 보간을 사용하는 것이 매우 바람직합니다.

- **비보간 모드**: 현재 픽셀의 위 및/또는 왼쪽으로 가장 가까운 픽셀을 복제합니다.
- **표준 보간 모드**: 적절한 인접 픽셀의 평균을 냅니다.
- **확장 보간 모드(권장)**: 알려지지 않은 구성 요소 값을 보간합니다. 이 모드는 개체 에지를 따라 나타날 수 있는 결함을 줄여줍니다.

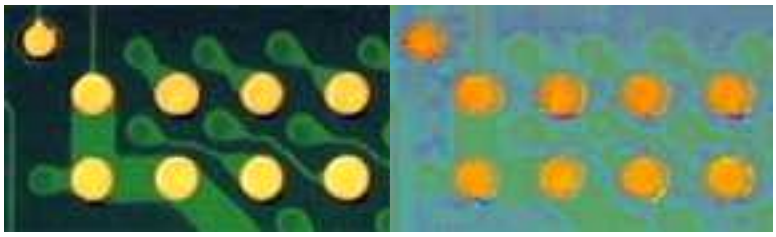


비보간(맨 위), 표준(왼쪽), 확장 보간 방식(오른쪽)으로 변환된 이미지

이득/오프셋용 LUT(컬러)

이미지의 세 성분에 각각 별도의 이득과 오프셋을 적용할 수 있습니다(대비 강조 변환). RGB 이미지는 대상 색 공간으로 변환하고, 이득과 오프셋을 적용한 다음, 다시 RGB로 변환해야 합니다.

- 혼합 표시에 적용하면, 세 이득 및 오프셋 모두 비슷한 방식으로 변화하게 됩니다.
- 루마/크로마 표시에 적용하면, 유채색 성분의 이득과 오프셋이 비슷한 방식으로 변화하게 됩니다.
- 강도/채도/색조 표시에 적용하면, 색조 성분에 이득과 오프셋을 적용하는 것이 의미가 없어집니다.



확장된 채도/균일한 명도

참고: *대비 강조 기능을 사용하여 일정한 성분을 균일화할 수 있으며, 일부 성분에 대해 이득을 0으로 설정하면 모든 픽셀을 해당 성분의 오프셋 값으로 설정하는 것과 같은 결과를 얻을 수 있습니다.*

컬러 교정용 LUT

이미지 캡처 체인으로 인해 발생된 컬러 왜곡은 이미지의 샘플 컬러와 실제 값을 비교하여 정정할 수 있습니다. 여기에 IT8과 같은 교정 컬러 차트가 필요합니다.

- 샘플 컬러는 적절한 ROI 내의 `PixelAverage`를 사용한 평균 컬러입니다..
- 실제 컬러 값은 XYZ 컬러 시스템으로 지정됩니다. 레퍼런스 컬러가 XYZ 좌표로 지정된 경우에도, 교정 대상 이미지에 RGB 정보가 포함되어 있어야 합니다.

교정 변환은 하나, 셋 또는 네 개의 레퍼런스 컬러를 기준으로 실행할 수 있습니다. 첫 번째 사례에서는 세 컬러 성분에 대한 이득 조정으로 교정이 이루어집니다. 두 번째와 세 번째 사례에서는 선형 또는 아핀 변환이 사용됩니다.

컬러 밸런스용 LUT

감마 수정과 화이트 밸런스를 변경하여 컬러 이미지를 개선할 수 있습니다.

WhiteBalance를 사용하여 조회 테이블을 설정하고 Transform(변환)을 사용하여 이를 일련의 이미지에 적용함으로써 이러한 효과를 효율적으로 수정할 수 있습니다. LUT는 한 번만 준비하면 됩니다(비연계 컬러 변환을 구현합니다).

감마 사전 보정

대부분의 컬러 카메라는 디스플레이 장치(예를 들어 TV 모니터)의 비선형 응답에 대응하는 감마 사전 보정 기능을 사용합니다.

감마 사전 보정은 처리 후에 사용해야 하며, 처리 전에 사용하면 비선형이 개입되어 결과가 변경될 것이기 때문입니다.

사전 보정 프로세스는 이미지가 디스플레이에 올바르게 표시될 수 있도록 신호의 반전 변환에 적용됩니다. 사용 중인 비디오 표준에 따라 세 가지 사전 정의 감마 값을 사용할 수 있습니다.

비디오 표준	감마 값	EasyColor 속성
NTSC	1/2.2	CompensateNtscGamma
PAL	1/2.8	CompensatePalGamma
SMPTE	0.45	CompensateSmpteGamma

참고: *사전 보정 취소 및 순수 사전 보정은 서로 반비례하는 지수에 대응합니다.*

감마 사전 보정 취소

대부분의 컬러 카메라에는 내장 감마 사전 보정 기능이 있으며 이를 끌 수 있습니다. 이 기능을 원하지 않지만 끌 수 없는 경우, 직접 감마 변환을 적용하여 그 효과를 취소할 수 있습니다. 다음은 이러한 목적으로 사용할 수 있는 사전 정의 감마 값입니다.

비디오 표준	감마 값	EasyColor 속성
NTSC	2.2	NtscGamma
PAL	2.8	PalGamma
SMPTE	1/0.45	SmpteGamma

화이트 밸런스

카메라가 컬러 불균형 상태 즉, 세 컬러 채널의 이득이 불일치하거나, 광원(빛의 근원)이 완전한 흰색이 아닐 경우가 있을 수 있습니다. 이러한 경우, 흰색 영역이 불포화 컬러로 나타납니다. 화이트 밸런스 정정 기능은 흰색 픽셀의 성분이 균일해지도록 세 독립적 이득을 자동으로 조정합니다. 이는 화이트 밸런스 보정 단계가 필요하다면, 그 과정에서 흰색 표면을 카메라에 비춰 해당 컬러 성분을 측정해야 함을 의미합니다. PixelAverage를 이 목적으로 사용할 수 있습니다.

원시(Raw) 이미지, 화이트 밸런스 및 감마 사전 보정이 적용된 이미지

3.3. EasyImage - 이미지 통계 계산하기

EasyObject 통계는 이미지 내의 개체와 연관됩니다.

EasyImage statistics는 전체 이미지(전역 조명/대비, 채도, 개체의 존재 또는 부재 여부)와 연관됩니다.

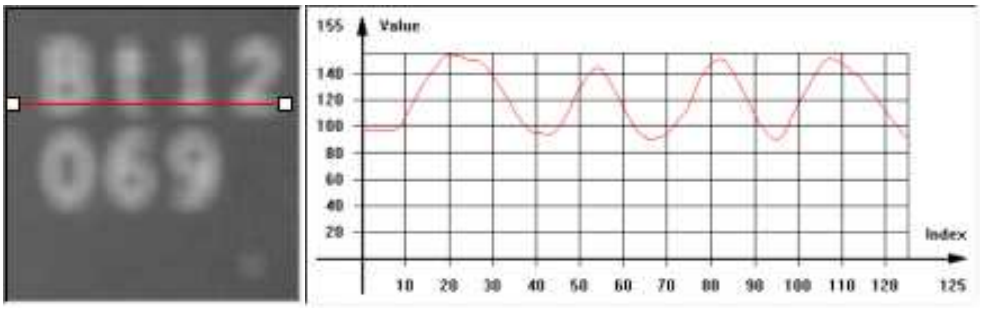
슬라이딩 윈도우(회색조 값의 평균 또는 표준 편차의 새로운 이미지 생성)

회색조 이미지의 평균 또는 표준 편차를 예를 들어, 모든 픽셀에 중앙 정렬된 사각형 창의 모든 위치에 대해 계산되는 슬라이딩 윈도우 내에서 계산할 수 있습니다. 윈도우 크기는 임의입니다.

참고: *이 함수의 계산 시간은 윈도우 크기에 따라 결정되지 않습니다.*

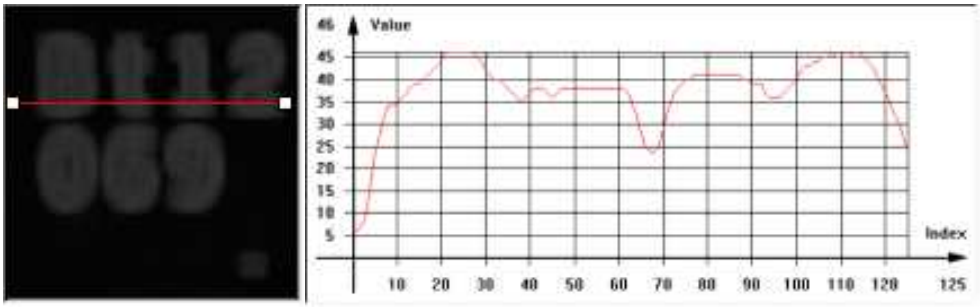
작업 결과는 다른 이미지로 표시됩니다.

국소 평균, `EasyImage::LocalAverage`는 강력한 저역 통과 필터링에 해당합니다.



슬라이딩 윈도우 평균

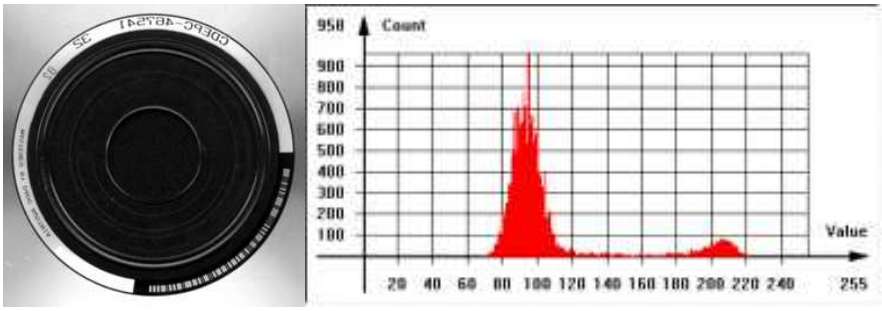
국소 표준 편차, `EasyImage::LocalDeviation`는 노이즈 또는 텍스처가 있는 영역과 같이 내용물 빈도가 높은 영역을 강조합니다.



슬라이딩 윈도우 표준 편차

히스토그램 계산 및 분석(및 LUT 생성)

히스토그램은 이미지의 통계적 요약으로, 이미지 내에 있는 모든 회색조 값의 발생 횟수를 보여주며, 그 형상으로 이미지의 특성을 알 수 있습니다. 예를 들어, 히스토그램 곡선에 있는 피크는 이미지의 주조색에 해당합니다. 히스토그램이 바이모달일 경우, 어두운 값의 커다란 피크는 배경, 밝은 값에 포함된 작은 피크에 해당합니다.



일반적인 이미지 히스토그램

히스토그램 계산

`EasyImage::Histogram`은 이미지의 히스토그램을 계산합니다. 이 함수에는 입력 마스크 인수가 있으며,
플렉시블 마스크를 지원합니다.
또한 BW8, BW16, BW32 소스 이미지를 지원합니다.

이미지의 누적 히스토그램을 계산할 수 있습니다. 즉, `EasyImage::Histogram`에 이어 `EasyImage::CumulateHistogram`을 호출하여 지정된 임계값 아래의 픽셀 수 등과 같이 이미지의 누적 히스토그램을 계산할 수 있습니다.

히스토그램 분석

`EasyImage::AnalyseHistogram` 및
`EasyImage::AnalyseHistogramBW16`은 다음과 같은 통계 및 임계값을 제공합니다.

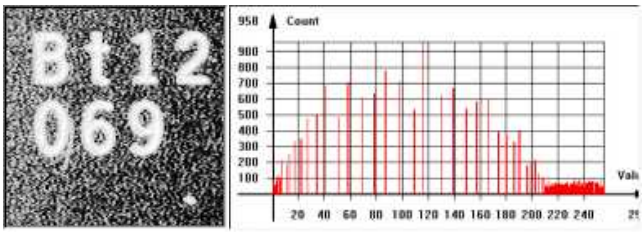
- 총 픽셀 수.
- 최소 및 최대 픽셀 값(회색조 범위).

- 픽셀 값의 평균 및 표준 편차.
- 가장 빈번한 픽셀의 값과 빈도.
- 가장 덜 빈번한 픽셀의 값과 빈도.

히스토그램 균등화

EasyImage::Equalize는 히스토그램이 전체 다이나믹 레인지를 가능한 균일하게 채우도록 회색조를 재매핑합니다.

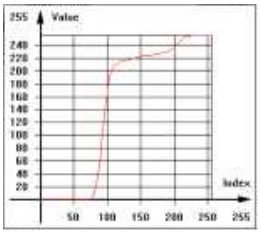
이는 이미지 대비를 극대화하거나, 어두운 부분의 이미지 세부 정보를 다수 밝히는 데 유용할 수 있습니다.



균등화된 이미지와 히스토그램

조회 테이블 설정

EasyImage::SetupEqualize는 히스토그램 및 LUT 벡터를 분명하게 다룰 수 있도록 LUT를 생성합니다. 이미지 히스토그램은 다른 목적(예: 통계)으로 보관하고, 균등화 LUT는 계속 다른 이미지에 적용하는 것이 더 효율적일 수 있습니다.



균등화 조회 테이블

이미지 초점

지정된 이미지에서 EasyImage::Focusing 수량이 최대라면 선명한 초점을 확보할 수 있습니다. 서로 다른 초점의 여러 이미지를 사용하는 경우 "자동 초점" 시스템을 기반으로 이 함수를 여러 번 호출해야 합니다.

EasyImage::Focusing은 이미지의 전체 그래디언트 에너지를 계산합니다. 그런 다음 이미지의 초점을 측정하는 데 이 그래디언트를 사용할 수 있습니다.

이미지의 그래디언트는 이미지에 존재하는 구조의 가장자리를 보여 주며, 이미지의 초점이 잘 맞으면 강한 값을, 그렇지 않으면 더 약한 값을 표시합니다.

이미지의 전체 그래디언트 에너지를 계산하려면 다음을 수행하십시오Open eVision:

a. 가로 및 세로 그라디언트 이미지의 픽셀 값을 제곱합니다.

b. 두 이미지에 대한 제곱된 픽셀 값을 평균합니다.

c. 평균을 합산합니다.

d. 결과 값의 제곱근을 취합니다.

이미지의 초점이 잘 맞으면 결과 값이 최대가 됩니다.

**(절대값) 수평 및 수직 그래디언트가 있는 잘 초점을 맞춘 이미지.
그래디언트는 강한 값을 가진 구조의 가장자리를 보여줍니다. 이 이미지의 전체 그래디언트 에너지는 17.9입니다.**

**(절대값) 수평 및 수직 그래디언트가 있는 초점이 맞지 않은 이미지.
그라디언트는 구조의 가장자리가 약한 값으로 표시됩니다. 이 이미지의 전체 그래디언트 에너지는 7.9입니다.**

EasyImage 통계 함수

면적(임계값 위/해당/사이의 픽셀 수)

- `EasyImage::Area`는 임계값 위의(또는 해당하는) 값을 가진 픽셀 수를 셉니다.
- `EasyImage::AreaDoubleThreshold`는 두 임계값 사이의(또는 해당하는) 값으로 구성된 픽셀 수를 셉니다.

바이너리 및 가중 모멘트(개체 위치 및 범위)

- `EasyImage::BinaryMoments`는 이진화 이미지에서 0차, 1차, 2차 모멘트를 계산합니다. 예를 들어, 임계값 위 또는 동일한 값의 픽셀에 대한 단위 가중치이며, 그렇지 않으면 0입니다. 이는 개체 위치 및 범위와 같은 정보를 제공합니다.
- `EasyImage::WeightedMoments`는 회색조 이미지에서 0차, 1차, 2차, 3차 또는 4차 가중 모멘트를 계산합니다. 픽셀의 가중치는 회색조 값입니다. 이는 개체 위치 및 범위와 같은 정보를 제공합니다.

무게중심(임계값 위/해당 평균 픽셀 좌표)

- `EasyImage::GravityCenter`는 이미지의 무게중심 좌표 즉, 임계값 위의(또는 해당하는)

픽셀의 평균 좌표를 계산합니다.

픽셀 개수(두 임계값 사이)

- `EasyImage::PixelCount`는 두 임계값으로 분리된 세 값 클래스에 포함된 픽셀 수를 셉니다.

최소, 최대, 평균 회색조 값

- `EasyImage::PixelMax`는 이미지 내의 최대 회색조 값을 계산합니다.
- `EasyImage::PixelMin`은 이미지 내의 최소 회색조 값을 계산합니다.
- `EasyImage::PixelAverage`는 회색조 또는 컬러 이미지 내의 평균 픽셀 값을 계산합니다. 컬러 이미지의 경우, 세 픽셀 컬러 성분의 평균, 성분의 분산, 성분 쌍 사이의 공분산을 계산합니다.

평균, 분산, 표준 편차

- `EasyImage::PixelStat`은 최소, 최대, 평균 회색조 값을 계산합니다.
- `EasyImage::PixelVariance`는 픽셀 값의 평균과 분산을 계산합니다.
- `EasyImage::PixelStdDev`는 픽셀 값의 평균과 표준 편차를 계산합니다. 컬러 이미지의 경우, 픽셀 성분 값 쌍의 표준 편차와 상관관계 계수(표준 편차의 곱에 대한 공분산)를 계산합니다.

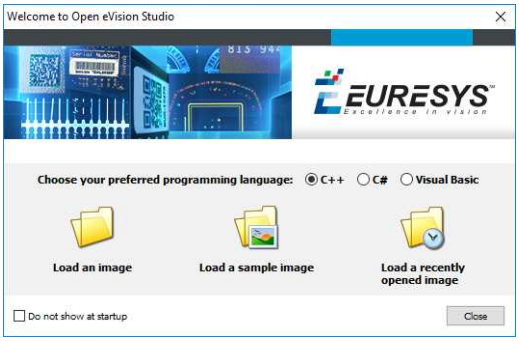
두 이미지를 비교한 결과에서 서로 다른 픽셀의 수

- `EasyImage::PixelCompare`는 두 이미지 사이에서 서로 다른 픽셀의 수를 셉니다.

4. Open eVision Studio 사용하기

4.1. 프로그래밍 언어를 선택하십시오.

처음 시작할 때 Open eVision Studio 다음과 같은 웰컴스크린을 표시합니다.



1. 프로그래밍 언어를 선택하십시오.

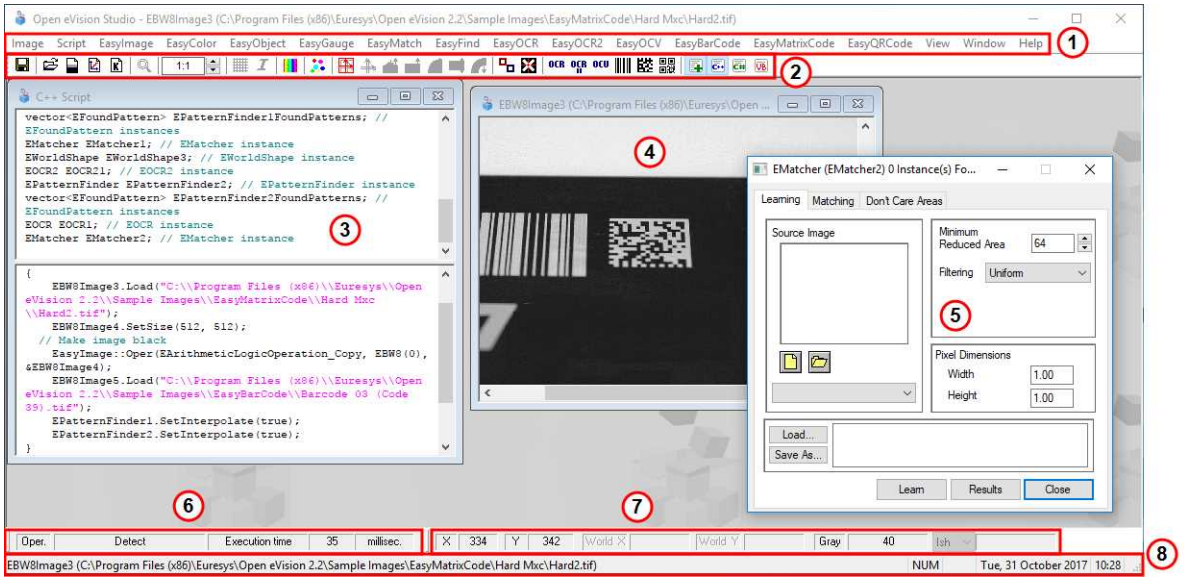
선택한 것을 저장하고 다음 시작 시에는 사용자의 프로그래밍 언어를 자동으로 선택합니다 Open eVision Studio.

> **참고:** *프로그래밍 언어를 바꾸면 스크립트 윈도우에 있는 어떤 스크립트든지 자동으로 삭제되고 윈도우 내용이 초기화됩니다.*

2. 이미 불러들인 한 개 혹은 몇 개의 이미지를 나중에 처리하기 위해서 **불러오기** 버튼 중 하나를 클릭하십시오.
3. 다음 시작시 웰컴 스크린을 보고 싶지 않으면 **시작할 때 보지 않기** 박스를 체크하십시오. Open eVision Studio.

*웰컴 스크린을 보려면 언제든지 **도움**> **웰컴 스크린** 메뉴로 가서 설정을 변경하십시오.*

4.2. 인터페이스 둘러보기



Open eVision Studio 그래픽 유저 인터페이스 (GUI) 는 다음과 같이 이루어져 있습니다:

1. *메인 메뉴 바* 에서 모든 라이브러리의 기능과 툴을 이용할 수 있습니다.

Open eVision Studio 라이선스가 필요하지 않으며 모든 라이브러리를 시험해볼 수 있습니다. 물론 Open eVision Studio 에서 사용자의 애플리케이션에 코드를 복사하였지만 필요한 라이센스가 없으면 실행시 "라이센스 없음" 오류 메세지를 보여줍니다.

2. *주요 도구 바* 는 이미지, 모양, 게이지, 바코드, 매트릭스 코드와 같은 주요 Open eVision 개체에 간편하게 액세스 할 수 있습니다.
3. *스크립트 윈도우* 는 Open eVision Studio에서 수행한 작업에 해당하는 코드를 선택한 프로그래밍 언어로 표시합니다. 언제든지 이 코드를 자신의 응용 프로그램에 저장하거나 복사할 수 있습니다.
4. *이미지 윈도우* 에는 라이브러리 및 도구를 사용하여 처리 할 수 있는 오픈 이미지가 표시됩니다.
5. *도구 윈도우* 를 사용하여 사용 가능한 모든 도구를 쉽게 구성할 수 있습니다. 해당 설정이 스크립트 윈도우에 자동으로 추가되어 쉽게 재사용할 수 있습니다.

대부분의 도구 창은 유동적이어서 화면 크기를 보다 잘 활용하기 위해 Open eVision Studio 기본 창 밖으로 쉽게 이동할 수 있습니다.

6. *실행 시간 막대* 는 컴퓨터에서 선택한 기능 (밀리 초 또는 마이크로 초 단위로 측정) 실행에 소요된 정확한 시간을 표시합니다. 이 정확한 측정은 애플리케이션의 성능을 평가하는 데에 도움이 됩니다.
7. *색상 도구 모음* 에는 이미지 커서의 X 및 Y 좌표와 해당 픽셀 값과 같은 현재 정보가 표시됩니다.
8. *상태 표시 줄* 에는 활성 이미지 파일 경로와 같은 애플리케이션에 대한 일반 정보가 표시됩니다.

4.3. 이미지에서 도구를 실행하기

1단계 : 도구 구성하기

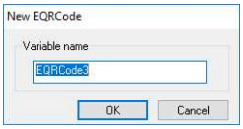
일반적으로 첫 번째 단계는, Open eVision Studio을 사용할 때, 이미지에 사용할 라이브러리와 도구를 선택합니다.

그렇게 하려면 :

1. 기본 메뉴 표시 줄에서 사용할 라이브러리를 클릭하십시오.
2. 사용할 도구를 클릭하십시오.

*모든 라이브러리 (EasyImage, EasyColor 및 EasyGauge 제외)는 **New Xxx Tool**이라는 하나의 도구만 표시합니다. 이 라이브러리 중 일부는 추가 기능도 제공합니다.*

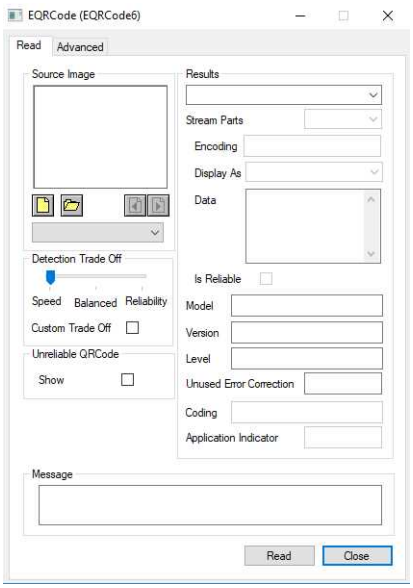
3. 대화 상자에서 자동으로 생성되고 처리 결과가 포함될 변수의 **Variable name** 을 입력하십시오.



EasyQRCode에 대한 변수 생성 대화 상자의 예

4. **Ok**를 클릭하십시오.

선택한 도구 대화 상자가 열립니다.



EasyQRCode에 대한 변수 생성 대화 상자의 예

다음 단계는 "2단계 : 이미지 열기" 아래.

2단계 : 이미지 열기

라이브러리와 도구를 선택했으면 이 도구를 적용할 이미지를 열어야 합니다.

선택한 도구 대화 상자의 **소스 이미지** 영역에서 다음을 수행합니다.

1. 이미지 열기:
   - Click on the **이미지 열기** 버튼을 클릭하고 컴퓨터에서 하나 이상의 이미지 (SHIFT 및 CTRL 사용)를 선택하십시오.
   - 또는 이미 드롭 다운 목록에서 열려있는 이미지 중 하나 (또는 ROI가 있는 경우) 중 하나를 선택하십시오.

   참고: *적절한 파일 형식 (JPG, PNG, TIFF 또는 BMP) 및 라이브러리에 따라 8비트 및 / 또는 24 비트의 이미지만 선택할 수 있습니다.*

2. 이미지를 여러 개 선택한 경우 **Load Previous** 또는 **Load Next** 버튼으로 이미지를 활성화하십시오.

이 도구는 로드된 이미지에 자동으로 적용되며,이 단계에서 도구 기본 설정에 따라 결과가 표시됩니다.

다음 단계는 "3단계 : ROI 관리하기" 아래.

3단계 : ROI 관리하기

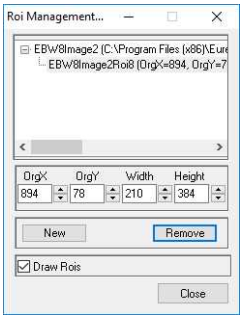
몇 가지 경우에, 대부분 처리 시간을 줄이거 나 읽으려는 대상을 단일화하기 위해서는, 전체 이미지를 처리하는 것이 아니라이 이미지의 하나 또는 여러 개의 잘 정의 된 사각형 부분 또는 ROI (관심 영역)를 처리하기를 원하지 않을 것입니다.

Open eVision에서 ROI는 이미지에 첨부되어 있으며 부모 이미지가 있는 동안에만 존재합니다.

ROI 생성하기

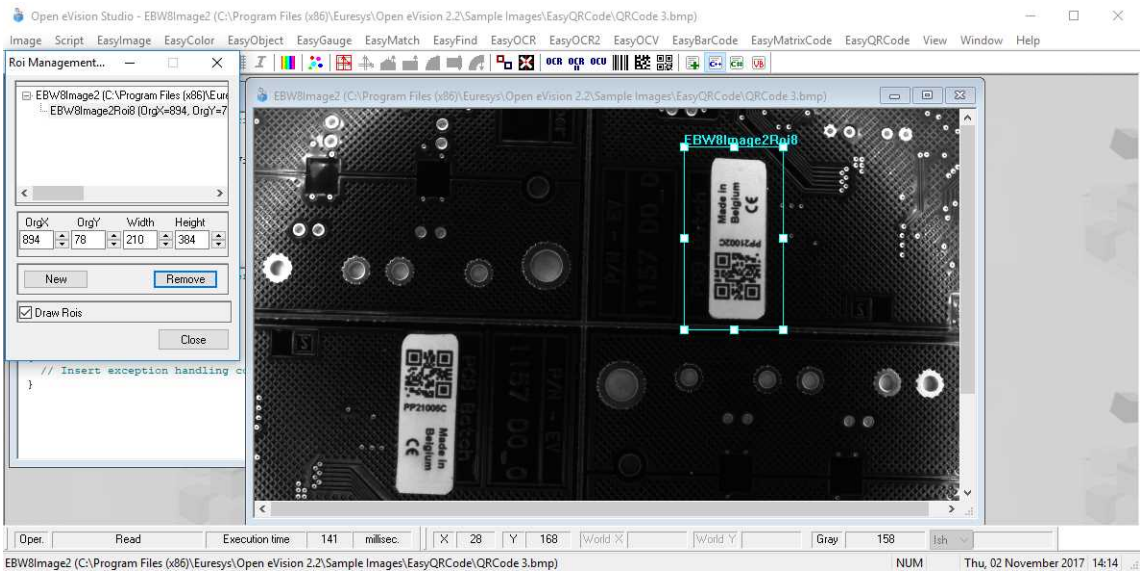
1. 이미지 열기:
   - 이미지가 이미 열려 있으면 해당 이미지 창을 활성화하십시오.
   - 이미지가 아직 열리지 않은 경우 기본 메뉴로 이동하십시오: 열기 위해서 **Image** > **Open...**
2. ROI를 만들려면 기본 메뉴로 이동하십시오: **Image** > **ROI Management...**.

ROI Management 윈도우는 아래 보이는 대로 표시됩니다.



3. 트리에서 이미지를 선택하십시오.
4. **New** 버튼을 클릭하십시오.
5. 대화 상자에서 새 RIO를 위한 **Variable name** 를 넣으십시오.

ROI는 아래 그림과 같이 이미지에 색상 사각형으로 표시됩니다.



6. ROI 모서리와 측면 핸들을 끌어 필요한 위치로 이동하십시오.
7. **ROI Management** 윈도우를 닫기 위해 **Close** 버튼을 클릭하십시오.

다음 단계는 "4 단계 : 도구 구성하기" 다음 페이지.

ROI 관리하기

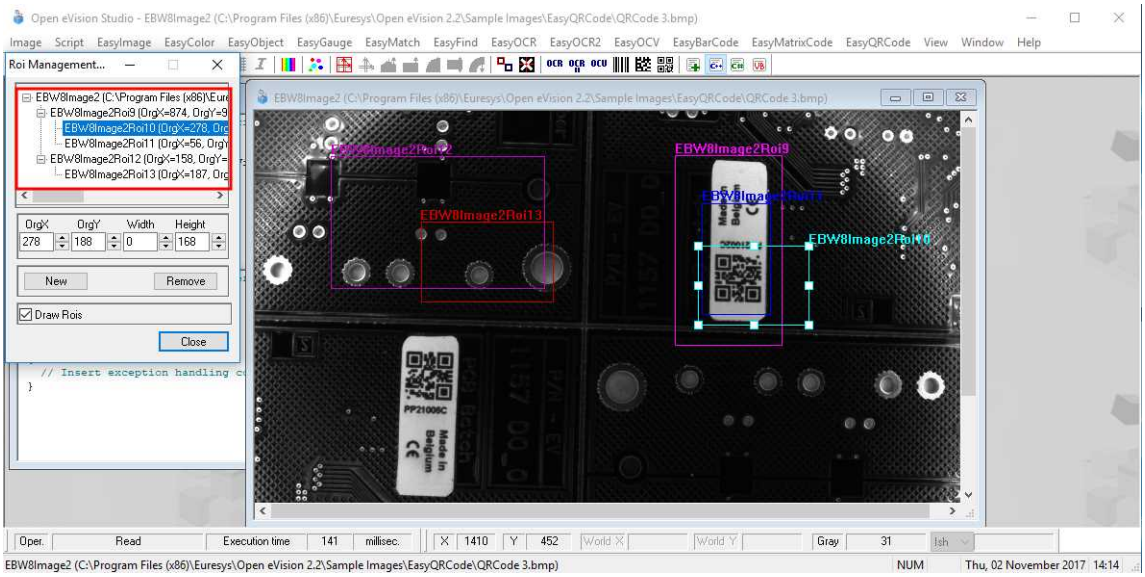
ROI를 추가, 변경 및 제거할 수 있습니다.

이미지는 여러 개의 ROI를 가질 수 있습니다. 각 ROI는 이미지에 직접 첨부 될 수 있습니다 (즉, 위치가 이미지와 관련됨) 또는 다른 ROI (이 위치가이 '부모' ROI와 관련이 있음을 의미함)에 첨부할 수 있습니다.

1. ROI를 관리하려면, 기본 메뉴로 이동하십시오: **Image** > **ROI Management...**.

ROI Management 윈도우가 아래와 같이 ROI 관계 트리와 함께 표시됩니다.

Draw Rois 상자가 선택되면 모든 ROI가 이미지에 다른 색상으로 표시됩니다.



2. ROI 관계 트리에서 ROI를 선택하십시오.
3. ROI 모서리와 측면 핸들을 드래그하여 선택한 ROI의 위치와 크기 (해당하는 경우 모든 ROI의 위치)를 변경하십시오.
4. **New** 클릭하면 선택한 ROI에 첨부 된 새로운 ROI를 추가 할 수 있습니다.

ROI를 이미지에 직접 첨부하려면 ROI 관계 트리의 상단에서 이미지를 선택하십시오.

5. 선택한 ROI (있는 경우 모든 해당 ROI가 첨부됩니다)를 삭제하려면 **Remove** 버튼을 클릭하십시오.
6. **ROI Management** 윈도우를 닫기위해 **Close** 버튼을 클릭하십시오.

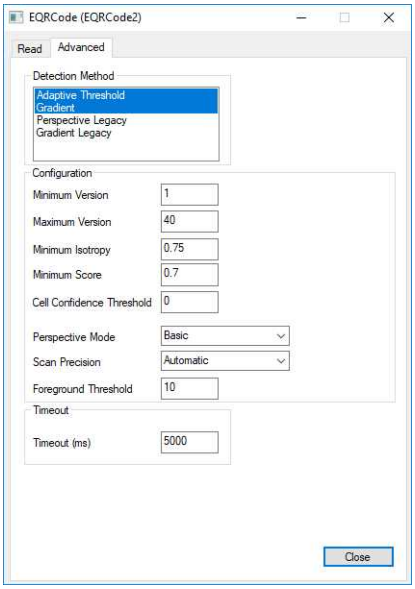
4 단계 : 도구 구성하기

일단 ROI를 작성했다면 ROI를 포함하여 이미지가 준비되면 도구를 구성해야 합니다.

도구 윈도우에서:

1. 여러 탭을여십시오.

새 도구를 생성하면 모든 매개 변수가 기본값으로 설정됩니다.



EasyQRCode 도구의 매개 변수 탭 예제

2. 각 탭에서 원하는대로 매개 변수의 값을 설정하십시오.

매개 변수, 기능 및 기본값에 대한 자세한 내용은 "기능 안내서"및 "레퍼런스 매뉴얼"을 참조하십시오.

학습 또는 게이지 사용과 같은 특정 작업에 대해서는 "기능 안내서"를 참조하십시오.

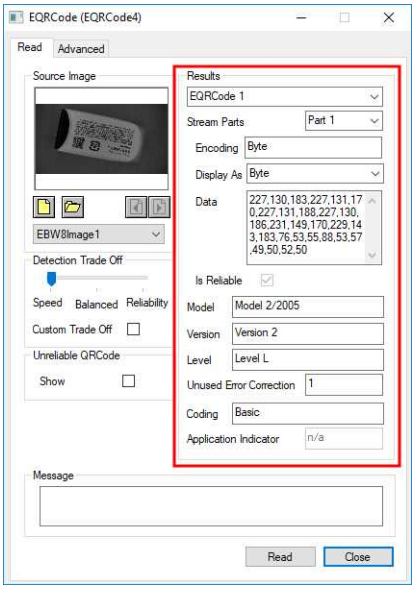
3. 다음 단계에서 설명한대로 도구를 실행하고 결과를 분석하십시오 "5 단계 : 도구 실행 및 실행 시간 확인하기" 아래.

5 단계 : 도구 실행 및 실행 시간 확인하기

도구 매개 변수가 설정되면 도구를 실행하고 원하는 경우 컴퓨터에서 실행 시간을 확인하십시오.

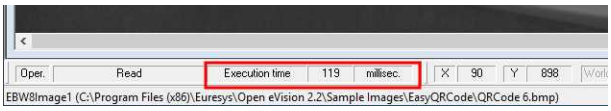
도구 윈도우에서:

1. 선택한 이미지에서 도구를 실행하려면 **Read**, **Detect**, **Results** 혹은 **Execute** 버튼을 클릭하십시오.(라이브러리 기능에 따라 다름)
2. 아래 그림과 같이 이미지와 결과 필드 또는 영역에서 결과를 확인하십시오.



QRCode를 읽은 결과의 예

3. 예상되는 결과가 없는 경우:
   - 매개 변수를 변경하십시오 (기본값으로 시작한 다음 한 번에 하나의 매개 변수만 변경하십시오).
   - 이미지가 충분하지 않은 경우, .설명 된대로 이미지를 강화시키십시오.
4. 기본 Open eVision Studio윈도우의 왼쪽 하단에 있는 실행 시간 막대에서 실행 시간을 확인하십시오.



실행 시간

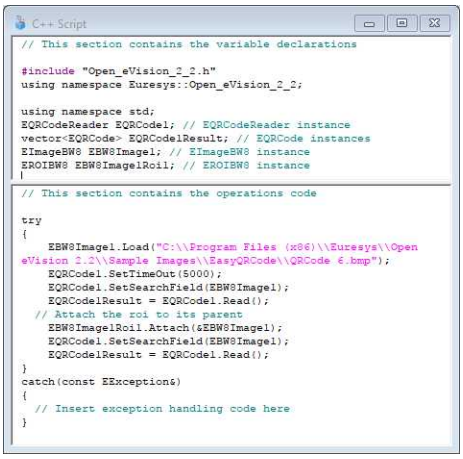
실행 시간은 컴퓨터에서 측정한 실제 처리 시간입니다. 그것은 컴퓨터 프로세서, 메모리, 운영 체제 및 실행시 프로세서 로드에 따라 다릅니다. 따라서 이 실행 시간은 실행마다 약간씩 다릅니다.

5. 보다 대표적인 실행 시간을 얻으려면 **Read**, **Detect**, **Results** 혹은 **Execute** 여러 번 클릭하고 평균 실행 시간을 계산하십시오.
6. 애플리케이션에서 실행 시간을 줄여야하는 경우 다음을 시도하십시오.
   - 도구 매개 변수를 변경하려면,
   - 이미지에 하나 이상의 ROI를 추가하려면,
   - 이미지 향상하기 위해

다음 단계는 "6단계: 생성된 콜드 사용하기" 다음 페이지.

6단계: 생성된 콜드 사용하기

기본적으로 Open eVision Studio 인터페이스에서 수행하는 모든 작업을 아래 그림과 같이 선택한 언어의 코드로 변환합니다.



도구 결과가 적합하면 이 생성된 코드를 저장하거나 복사하여 자신의 애플리케이션에서 사용할 수 있습니다.

애플리케이션에 코드 복사 및 붙여 넣기

스크립트 윈도우에서:

1. 복사할 코드 섹션을 선택하십시오.
2. 이 코드에서 오른쪽 클릭하고 메뉴에서 **복사** 를 클릭하십시오.
3. 개발 환경 도구로 가서 코드를 제자리에 붙여 넣으십시오.

코드를 저장하십시오.

1. **스크립트** 메뉴로 가십시오.
2. **다른 스크립트로 저장...**을 클릭하십시오.
3. 코드를 텍스트 파일로 저장하려면 파일 이름과 경로를 입력하십시오.

생성된 코드 관리하기

스크립트 메뉴에서:

- 프로그래밍 언어를 선택하십시오 (언어를 변경하면 스크립트 창 콘텐츠가 자동으로 삭제됩니다).
- **스크립트 코드 생성**을 활성화하거나 비활성화하십시오. 코드로 저장하지 않고 일부 작업을 수행하려면 이 옵션을 비활성화하십시오.

4.4. 이미지 전처리 및 저장

언제 이미지를 사전 처리해야합니까?

물론, 최적의 상황은 이미지를 쉽고 잘 처리 할 수 있도록 이미지 수집 시스템을 설정하여 Open eVision 도구를 원활하고 효율적으로 실행하는 것입니다.

이것이 가능하지 않거나 쉽게 할 수없는 경우 실행하고 싶은 Open eVision 도구를 위해서 이미지 또는 ROI를 사전 처리하여 강조하고 대비해 둘 수 있습니다.

다양한 사용 가능한 기능을 사용하여 이미지의 이득과 오프셋을 조정하고, 컨볼루션을 적용하고, 임계값을 조정하고, 이미지를 회전 및 화이트 밸런스 조정하고, 윤곽선을 강조할 수 있습니다... EasyImage 와 EasyColor 기능을 사용합니다.

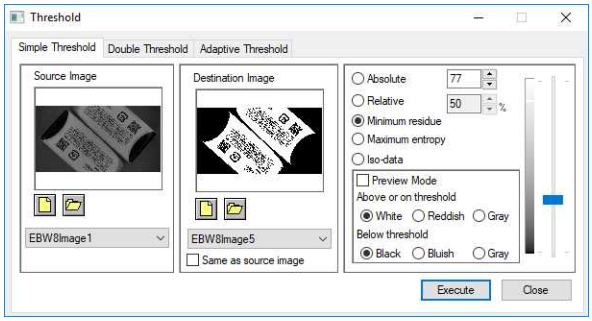
이미지 전처리

이미지 전처리와 실행중인 도구의 차이점은 전처리가 새로운 이미지를 생성하는 반면 도구는 주로 이미지를 변경하지 않고 이미지에서 추출 및 검색한다는 점입니다.

이미지나 ROI 전처리하기:

1. 기본 메뉴 표시 줄에서 사용하려는 라이브러리 (EasyImage 또는 EasyColor)를 클릭하십시오.
2. 사용할 기능을 클릭하십시오.

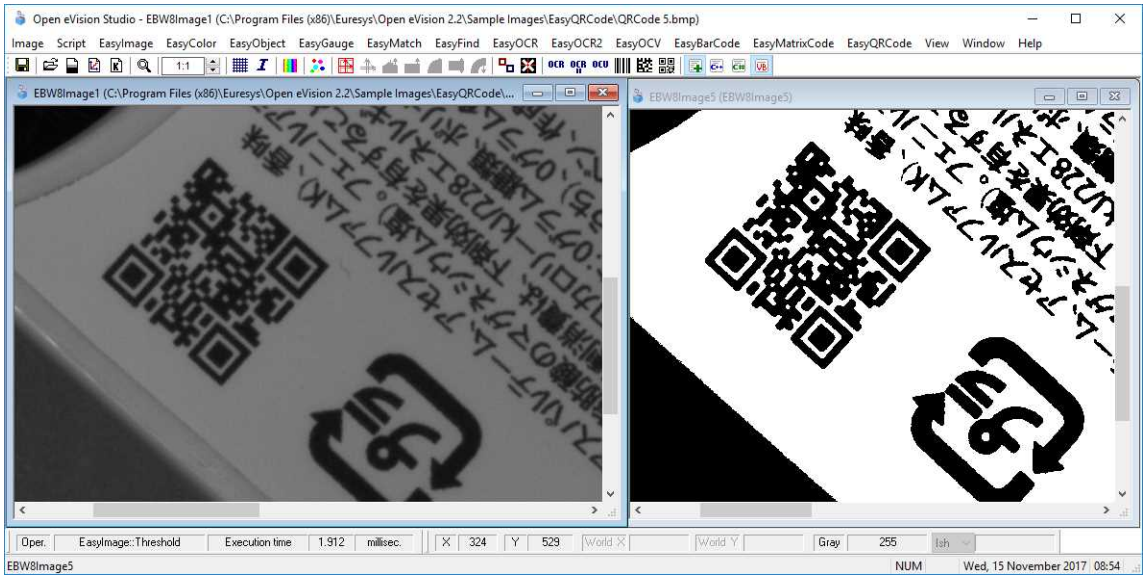
대부분의 기능 대화 상자는 아래 그림과 같이 2 개의 이미지 선택 영역과 매개 변수 설정 영역으로 비슷하게 되어 있습니다.



사전 처리 대화 상자의 예 (EasyImage와 임계값)

3. 선택한 기능에 여러 버전이 있는 경우 해당 탭을 엽니다.
4. **소스 이미지** 영역에서, 소스 이미지를 엽니다.("2단계 : 이미지 열기" 페이지80에서 설명한 대로).
5. **대상 이미지** 영역에서, 새로운 대상 이미지를 열거나 만드십시오.
6. 매개 변수를 설정하십시오.
7. **실행** 버튼을 클릭하십시오.

아래 그림과 같이 대상 이미지에서 전처리된 이미지를 볼 수 있습니다.



소스 이미지와 대상 이미지 (EasyImage와 임계값)

8. Open eVision Studio외의 다른 곳에서 대상 이미지를 사용하고 싶으면, 아래 설명한 대로 이미지를 저장하십시오.

이미지 저장하기

1. 저장하려는 이미지를 클릭하여 활성화하십시오.
2. 저장 메뉴를 열려면 다음 중 하나를 수행하십시오.
   - 이미지에서 마우스 오른쪽 버튼을 클릭하십시오.
   - 혹은 기본 메뉴> **이미지**를 여십시오.
3. **다른 이름으로 저장...**을 클릭하십시오.
4. 파일 형식을 선택하십시오.(JPEG, JPEG2000, PNG, TIFF 또는 비트맵)
5. 파일 이름을 적고 경로를 선택하십시오.
6. **저장** 버튼을 클릭하십시오.

5. Tutorials

5.1. EasyImage

Converting a Gray-Level Image into a Binary Image

"Thresholding" 페이지 110

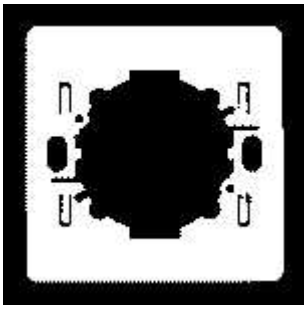
"Single Thresholding" 페이지 110 - "Double Thresholding" 페이지 110 - "Histogram-Based Single Thresholding" 페이지 111 - "Histogram-Based Double Thresholding" 페이지 111

Objective

Following this tutorial, you will learn how to use EasyImage to convert a gray-level source image into a binary destination image. Thresholding an image transforms all the gray pixels into black or white pixels, depending on whether they are below or above a specified threshold. Thresholding an image makes further analysis easier.

You'll need first to load an image (step 1). Then you'll set the thresholding parameters (step 2), and perform the conversion (step 3).

Gray-level source image



Black and white destination image, after thresholding

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Threshold**.
2. In the **Simple Threshold** tab, click the **Open** icon of the Source Image area, and load the image file `EasyMatch\Switch1.tif`.
3. Keep the default variable name for the new Image object, and click **OK**.

Step 2: Set the thresholding parameters

1. In the right area of the **Threshold** dialog box, move the slider to change the threshold, and see directly in the source image a preview of the result.
2. Select the **Minimum residue** option to set a pre-defined algorithm that finds automatically the right threshold.

Step 3: Perform the conversion

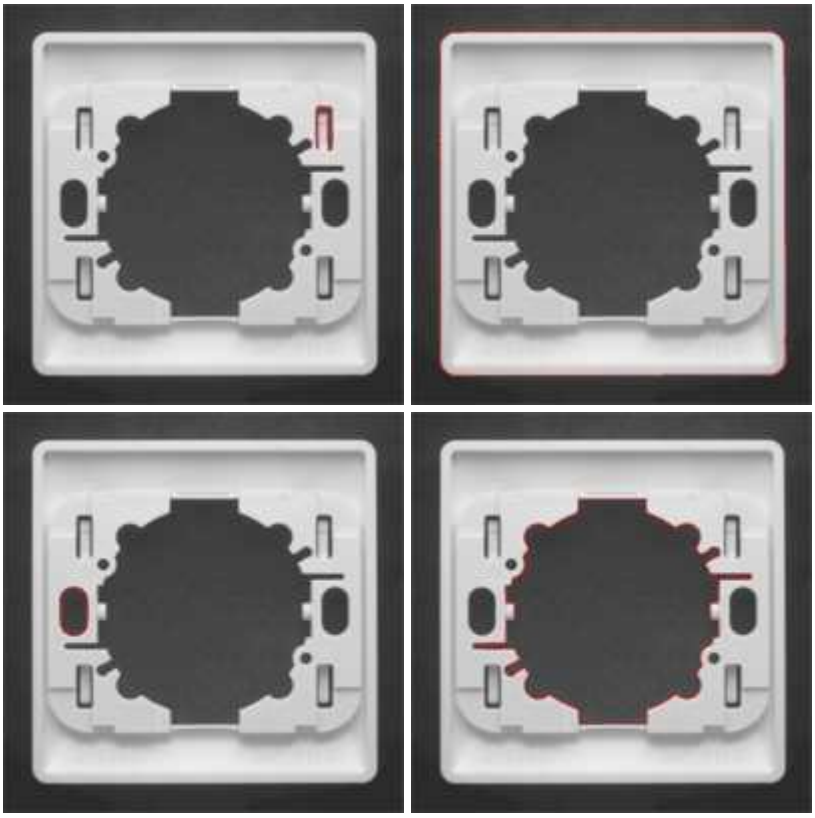
1. Click the **New** icon in the **Destination Image** area to create a new destination image.
2. Keep the default settings for the new Image object, and click **OK**.
3. In the **Threshold** dialog box, click **Execute** to perform the thresholding in the destination image.

Extracting an Object Contour

Objective

Following this tutorial, you will learn how to use EasyImage to trace an object outline in a gray-level image. The contour extraction allows you to get in a path vector all the points that constitute an object contour, just by clicking an edge of this object.

You'll need first to load an image (step 1) and set a vector that will contain all the contour points (step 2). Then you'll click an object edge, and the contour will be extracted automatically (step 3).



Contours are extracted from object edges

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Contour**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyMatch\Switch1.tif`.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Set the destination vector

1. Click the **New** icon in the Destination Vector area.
2. Keep the default settings and variable name for the new vector object.
3. Click **OK**.

Step 3: Extract the contour

1. When moving the cursor above the image, an arrow appears.
2. Move the arrow above an object edge, and click.

From this object edge, a contour is traced, and a red line appears around the object.

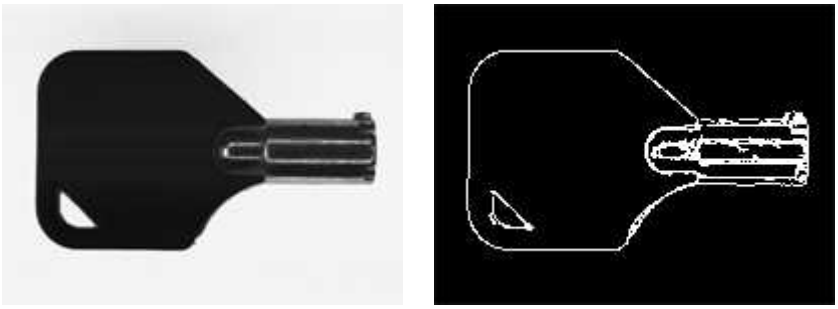
The destination vector is filled with the points constituting the contour.

Transforming a Gray-Level image into its Black and White Edges

Objective

Following this tutorial, you will learn how to use EasyImage to transform a gray-level image to a binary image, keeping only the edges detected in the image. The conversion uses the Canny edge detector algorithm.

You'll need to load a source image (step 1), and simply apply the Canny edge transformation (step 2).



Source image (left) and destination image after Canny edge transformation (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Canny Edge Detector**.
2. Keep the default variable name, and click **OK**.
3. Click the **Open** icon of the Source Image area, and load the image file `EasyImage\Key1.tif`.
4. Keep the default variable name, and click **OK**.

Step 2: Apply the Canny edge transformation

1. Click the **New** icon in the Destination Image area to create a new destination image.
2. Keep the default settings, and click **OK**.
3. In the canny edge detector dialog box, click **Apply** to perform the operation in the destination image.

Detecting the Corners of an Object Using Harris Corner Detector

Objective

Following this tutorial, you will learn how to use EasyImage to detect the corners of an object. The detection uses the Harris corner detector algorithm.

You'll need to load a source image (step 1), and simply apply the Harris corner detection (step 2).

Corners are detected in the source image

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Harris Corner Detector**.
2. Keep the default variable name, and click **OK**.
3. Click the **Open** icon of the Source Image area, and load the image file `EasyGauge\Bracket1.tif`.
4. Keep the default variable name, and click **OK**.

Step 2: Apply the Harris corner detection

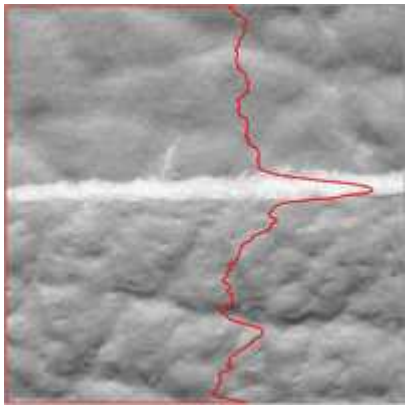
1. In the Harris corner detector dialog box, enter 2.3 for the Scale property.
2. Click **Apply** to perform corners detection.
3. Click **Results** to display the coordinates of all detected corners.
4. The **Columns** button allows you to display additional properties in the results list.

Detecting a Horizontal or Vertical Line Using Projection

Objective

Following this tutorial, you will learn how to use EasyImage to detect defects (horizontal/vertical line) in a gray-level image.

You'll need first to load a source image (step 1), set a vector (step 2), and then detect the defect (horizontal line) (step 3).



Defects can be detected using the image projection

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Projection**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyImage\Leather.bmp`.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Set the destination vector

1. Click the **New** icon in the Destination Vector area.
2. Select the BW32 option for the vector type, and click **OK**.

Step 3: Detect the defects

1. In the Image Projection dialog box, select the **column** button, and click **Execute** to perform the operation.
2. The resulting vector and the corresponding plot are displayed in the destination vector window. The graphical result also appears on the image. Each vector value is the sum of all pixels values across the corresponding horizontal row (or vertical column). By this mean, horizontal (or vertical) defects are easily detected.

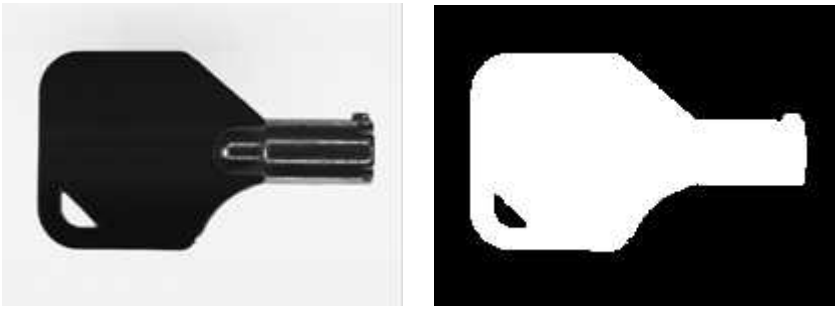
Creating a Flexible Mask

"Using Flexible Masks" 페이지 119

Objective

Following this tutorial, you will learn how to create a flexible mask from a source image, to restrict a future processing to an arbitrary-shaped do-care area.

Flexible masks can be created in any ways to build a bi-level image. Here, we will first load the source image (step 1), and then successively invert it, and threshold it (steps 2-3). The resulting image —the flexible mask— will be saved as a new image file (step 4). This new image file is a bi-level image. However, there are still black areas that need to be erased, before using the image as a flexible mask. You can use a third-party software, such as Paint, to clear the unwanted areas.



Source image (left) and flexible mask image (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Arithmetic & logic**.
2. Click the **Open** icon of the Source Image 0 area, and load the image file `EasyImage\Key1.tif`.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Invert the image

1. Click the **New** icon of the **Destination** area.
2. Keep the default settings for the new image object, and click **OK**.
3. In the **Operation** drop-down list, select **Invert**, and click **Execute**

Step 3: Threshold the image

1. From the main menu, click **EasyImage**, then **Threshold**.
2. In the Source Image area, select the inverted image from the drop-down list.
3. Click the **New** icon of the Destination area.
4. Keep the default settings for the new image object, and click **OK**.
5. Select the Absolute option, enter '46' as the threshold value, and click **Execute**.

Step 4: Save the flexible mask

1. Right-click in the destination image, and select **Save As...**.
2. Type a file name for the new flexible mask file. Finally, click **Save**.

The new image now is a bi-level image. However, there are still black areas that need to be erased, before using the image as a flexible mask. You can use a third-party software, such as Paint, to clear the unwanted areas.

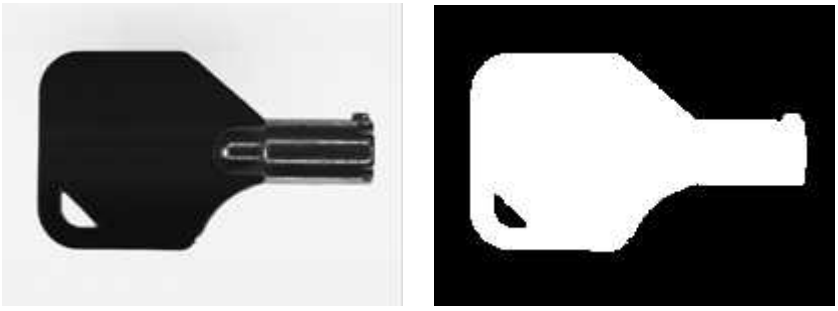
Computing Gray-Level Statistics Using a Flexible Mask

"Using Flexible Masks" 페이지 119

Objective

Following this tutorial, you will learn how to compute gray-level statistics on an arbitrary-shaped area only.

You'll need first to load a source image (step 1), and a flexible mask image (step 2). The mask image must be applied on the source image (step 3), to separate do-care areas (that must be considered) and don't-care areas (that should not be considered). Finally, the gray-level statistics are computed on the do-care area only (step 4).



Source image (left) and flexible mask image (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Image Statistics**, **Gray Scale**.
2. Click the **Open** icon of the Source Image area, and load the image file *EasyImage\Key1.tif*.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Load the flexible mask image

1. From the main menu, click **Image**, then **Open...**.
2. Load the image file *EasyImage\Mask2.bmp*.

3. Keep the default variable name for the new image, and click **OK**.

Step 3: Apply the flexible mask on the source image

1. In the Mask area of the **Gray Scale Image Pixels Statistics** dialog box, select the mask image from the drop-down list.
2. The source image preview in the dialog box shows (in red diagonal lines) the don't-care area, that is the area that will be not be considered when computing the gray-level statistics.

Step 4: Compute the gray-level statistics

1. Select the **Pixel Count** check-box.
2. Click **Execute**.

The results are displayed in the read-only fields below.

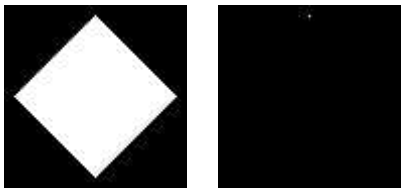
Detecting the Corners of an Object Using Hit-and-Miss Transform

"Hit-and-Miss Transform" 페이지 114

Objective

Following this tutorial, you will learn how to use EasyImage to detect top corners in an image, using the hit-and-miss transform.

You'll need to load a source image (step 1), set the kernel that represents a top corner (step 2), and then set a destination image and simply execute the hit-and-miss transform (step 3).



Source image (left) and top corner detected in the source image (white dot)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Hit And Miss**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyImage\Diamond.bmp`.
3. Keep the default variable name, and click **OK**.

Step 2: Set the hit-and-miss kernel

- In the **Hit And Miss** dialog box, set the kernel according to the following values:

	-1	0	1
-1	DontCare	Background	DontCare
0	Background	Foreground	Background
1	Foreground	Foreground	Foreground

Kernel that detects top corners

Step 3: Apply the hit-and-miss transform

1. Click the **New** icon of the **Destination Image** area.
2. Keep the default parameters and variable name, and click **OK**.
3. Click **Execute** to perform the operation.

The top corner (white dot) is detected.

4. Try with other kernel configurations to detect the other corners.

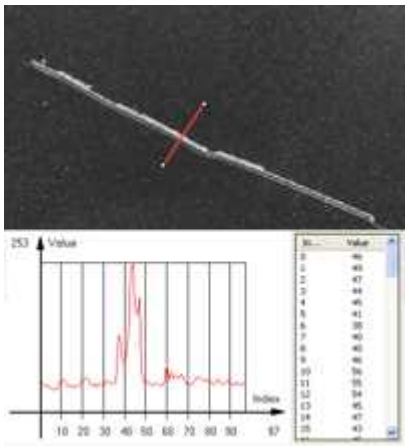
Extracting a Vector Using Profile Function

"Profile Sampling" 페이지 115

Objective

Following this tutorial, you will learn how to use EasyImage to detect scratches.

You'll need first to load an image (step 1), set a destination vector, and detect the scratches (step 2).



Scratches can be detected using a profile

Step 1: Load the source image

From the main menu, click **EasyImage**, then **Profile**.

1. Click the **Open** icon of the **Source Image** area, and load the image file `EasyImage\Plastic.tif`.
2. Keep the default variable name for the new image object, and click **OK**.

Step 2: Set the destination vector and detecting the scratches

1. Click the **New** icon in the Destination Vector area.
2. Select the BW8 option for the vector type, and click **OK**.
3. A profile appears on the image (red line segment). In the destination vector window, vector values correspond to pixels along the line segment.

The scratch is detected as a sharp deviation in the vector graph.

4. Using the mouse, drag the handles to move or resize the red line segment, and observe the plot evolution.

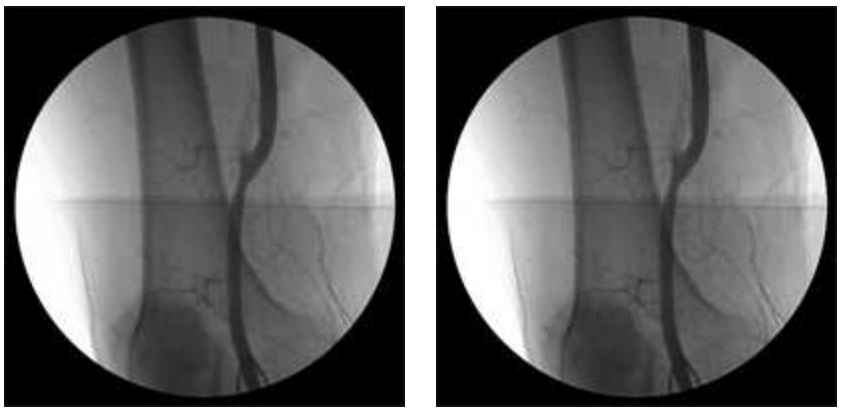
The sharp deviation appears whenever the line segment is placed across the scratch.

Enhancing an X-ray image

Objective

Following this tutorial, you will learn how to use EasyImage to enhance an X-ray image.

You'll need first to load an image (step 1), then define convolution parameters to enhance the image (step 2).



Source image (left) and enhanced image, after predefined and user-defined convolutions (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Convolution**.
2. Click the **Open** icon of the **Source Image** area, and load the image file `EasyImage\XRay.bmp`.
3. Keep the default variable name for the new image object, and click **OK**.
4. Click the **New** icon of the **Destination Image** area to create a new destination image.
5. Keep the default variable name and click **OK**.

Step 2: Set the convolution parameters

1. From the **Predefined kernels** drop-down list, select **Highpass2**, and click **Execute** to perform the operation.

The image is no longer blurred but the result is bad because the filter has revealed the noise of the source image. We need to create a new convolution kernel that will apply a softer high-pass filtering.

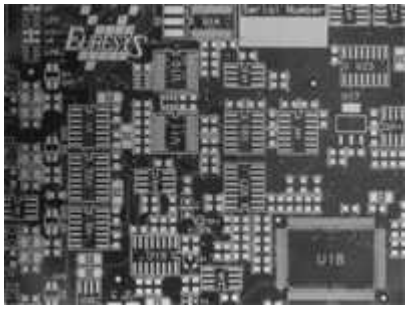
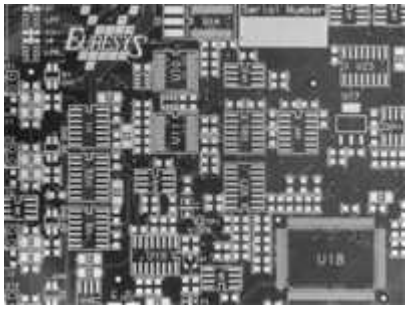
2. Click the **New** icon next to the **User defined kernels** drop-down list.
3. Keep the default dimension (3x3) and variable name, and click **OK**.
4. Enter the following kernel data from left to right and top to bottom: -1, -1, -1; -1, 15, -1; -1, -1, -1, and click **Apply**.
5. Click **Execute** in the Convolution dialog box to perform the operation. The image is much clearer now.

Correcting Non-Uniform Illumination

Objective

Following this tutorial, you will learn how to use EasyImage to correct non-uniform illumination in an image.

You'll need first to load an image (step 1), load a light reference image (step 2), and perform the correction (step 3).

Source image, with non-uniform illumination (left) and corrected image (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Uniformize**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyImage\Board (original).tif`.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Load the reference image

1. Click the **Open** icon of the **Light Reference** area, and load the image file `EasyImage\Board (light reference).tif`.

To obtain the light reference image, we used a white screen illuminated in the same condition as the board (original image).

2. Keep the default variable name for the new image object, and click **OK**.

Step 3: Perform the correction

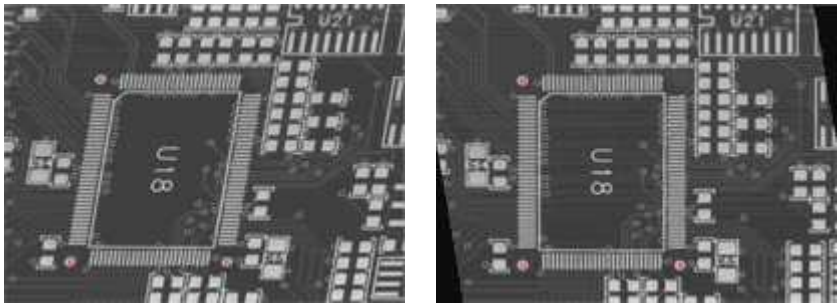
1. Click the **New** icon in the **Destination Image** area to create a new destination image.
2. Keep the default values and click **OK**.
3. Click **Execute** to perform the operation.
4. In both source and destination images, right-click and select **3D Rendering**.
5. In the new 3D windows, click and drag the mouse to rotate the view. Compare the profiles.

Correcting Shear Effect

Objective

Following this tutorial, you will learn how to use EasyImage to correct a shear effect in an image. The following image is taken by a line-scan camera. The camera sensor was misaligned, resulting in a so-called shear effect.

You'll need first to load an image (step 1), create a destination image (step 2), and then set pivots parameters to perform the correction (step 3).



Source image, with a shear effect (left) and corrected image (right)

Step 1: Load the source image

1. From the main menu, click **EasyImage**, then **Register**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyImage\Shear.tif`.
3. Keep the default variable name for the new image object, and click **OK**. Three pivots points appear in the image.

Step 2: Create a destination image

1. Click the **New** icon of the **Destination Image** area.
2. Enter '768' and '576' as image width and height, and click **OK** to accept the default name. Three pivots points appear in the image.

Step 3: Set the pivots parameters

1. In the source image, using the mouse, drag each pivot to the center of the fiducial marks (the dots around the U18 area).

Notice that the source pivots coordinates, in the Register dialog box, have changed accordingly.

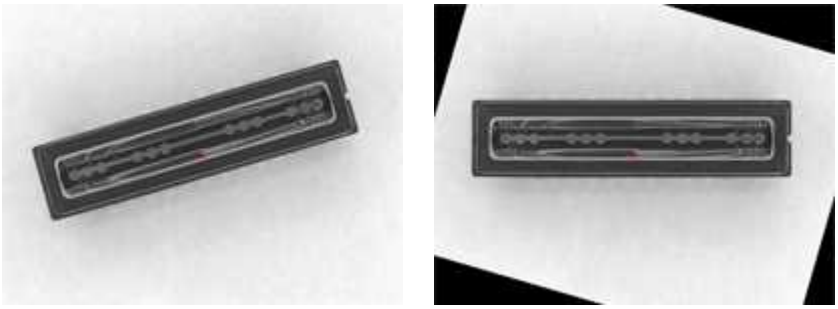
2. To correct the image, enter the following destination pivots coordinates:
   - X0: 170
   - Y0: 495
   - X1: 470
   - Y1: 495
   - X2: 170
   - Y2: 144
3. Click **Execute** to perform the operation.

Correcting Skew Effect

Objective

Following this tutorial, you will learn how to use EasyImage to correct skew effect in an image.

You'll need first to load an image (step 1), create a destination image (step 2), and then set a correction angle to perform the correction (step 3).



Source image, with a skew effect (left) and corrected image (right)

Step 1: Loading the source image

1. From the main menu, click **EasyImage**, then **Scale and Rotate**.
2. Click the **Open** icon of the **Source Image** area, and load the image file `EasyImage\CCD.tif`.
3. Keep the default variable name for the new image object, and click **OK**.

Step 2: Creating a destination image

1. Click the **New** icon of the **Destination Image** area.
2. Enter '768' and '576' as image width and height, and click **OK** to accept the default name.

Step 3: Setting the correction angle

1. Select the **Rotate** option, and enter -16.17 in the Angle (Deg) field. (To measure this rotation angle, refer to Measuring the rotation angle of an object.)
2. From the Interpolation bits drop-down list, select 8 bits to get a better result.
3. Click **Execute** to perform the operation.

5.2. EasyColor

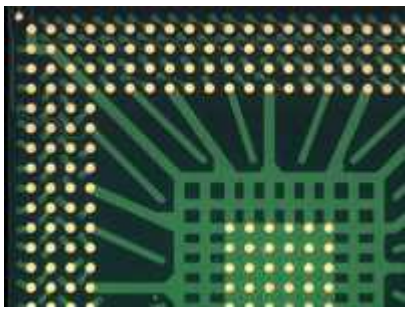
Performing Thresholding on Color Images

"Color Components" 페이지 120

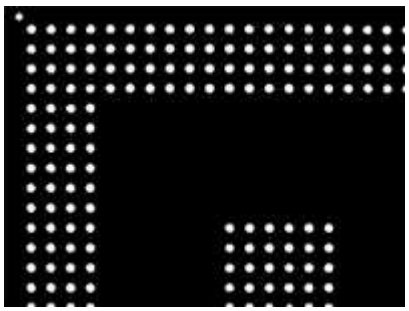
Objective

Following this tutorial, you will learn how to use EasyColor to segment a color source image, by setting a threshold value for each color component of the current color system. For example, to retrieve the solder pads on a PCB, you'll perform a color segmentation based on the golden pixels (H), with a loose discrimination on the brightness (L) and saturation (S), to eliminate surface and lighting effects.

You'll need first to load a color source image, create a destination image, and a color lookup table (steps 1-3). Then, you'll set the color system and tune each component tolerance to get a satisfying segmentation of the solder pads (step 4).



Source image



Thresholded image

Step 1: Load the source image

1. From the main menu, click **EasyColor**, then **Threshold**.
2. Click the **Open** icon of the **Source Image** area, and load the image file `EasyColor\BGA Substrate Color.jpg`.
3. Keep the default variable name for the new Image object, and click **OK**.
4. Disable the **Preview Mode** check-box to see the raw source image.

Step 2: Create a destination image

1. Click the **New** icon of the **Destination Image** area.
2. Keep the default variable name for the new Image object, and click **OK**.

Step 3: Create a color lookup table

1. Click the **New** icon of the **Color Lookup** area.
2. Keep the default variable name for the new color lookup object, and click **OK**.

Step 4: Perform the color segmentation

1. Select **LSH** from the **Color System** drop-down list.
2. In the source image, click in a golden pad. The pixel lightness, saturation and hue values are updated in the **Color Threshold** dialog box.
3. Adjust the tolerance of lightness and saturation to enlarge the range of thresholded pixels, until you get a satisfying segmentation in the destination image.

Performing Color Segmentation

"Color Components" 페이지 120

Objective

Following this tutorial, you will learn how to use EasyImage to perform color segmentation.

You'll need first to load an image (step 1), create a color look-up table (step 2), and perform the segmentation (step 3).

Source image

Segmented image

Step 1: Load the source image

1. From the main menu, click **EasyColor**, then **Threshold**.
2. Click the **Open** icon of the Source Image area, and load the image file `EasyColor\Pills.tif`.
3. Keep the default variable name for the new image object, and click **OK**.
4. Disable the **Preview Mode** check-box to see the raw source image.

Step 2: Create a color lookup table

1. Click the **New** icon of the Color Lookup area.
2. Keep the default variable name for the new color lookup object, and click **OK**.

Step 3: Perform the color segmentation

1. Select **LSH** from the **Color System** drop-down list.
2. In the source image, click in the center of a green pill. The pixel lightness, saturation and hue values are updated in the **Color Threshold** dialog box.
3. Increase the lightness tolerance up to 120. Increase the saturation tolerance up to 50.
4. Enable the **Preview Mode** check-box to see the result of the segmentation. If needed, click in the green pills to improve the result.
5. Click the **New** icon of the Destination Image area.
6. Keep the default settings for the new Image object, and click **OK**.
7. The new image is automatically thresholded. Clicking **Execute** will insert the corresponding code into the script windows.

6. Code Snippets

6.1. Basic Types

Loading and Saving Images

```
////////////////////////////////////////////////////////////
// This code snippet shows how to load and save an image. //
////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// Load an image file
srcImage.Load("mySourceImage.bmp");
```

```
// ...
```

```
// Save the destination image into a file
dstImage.Save("myDestImage.bmp");
```

```
// Save the destination image into a jpeg file
// The default compression quality is 75
dstImage.Save("myDestImage.jpg");
```

```
// Save the destination image into a jpeg file
// set the compression quality to 50
dstImage.SaveJpeg("myDestImage50.jpg", 50);
```

Interfacing Third-Party Images

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to link an Open eVision image //
// to an externally allocated buffer. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
```

```
// Size of the third-party image
int sizeX;
int sizeY;
```

```
//Pointer to the third-party image buffer
EBW8* imgPtr;
```

```
// ...
```

```
// Link the Open eVision image to the third-party image
// Assuming the corresponding buffer is aligned on 4 bytes
srcImage.SetImagePtr(sizeX, sizeY, imgPtr);
```

Retrieving Pixel Values

```
////////////////////////////////////////////////////////////
```

```
// This code snippet shows the recommended method (fastest)  //
// to access the pixel values in a BW8 image                 //
///////////////////////////////////////////////////////////////
```

```
EImageBW8 img;
```

```
OEV_UINT8* pixelPtr;
OEV_UINT8* rowPtr;
OEV_UINT8  pixelValue;
OEV_UINT32 rowPitch;
OEV_UINT32 x, y;
```

```
rowPtr = reinterpret_cast <OEV_UINT8*>(img.GetImagePtr());
rowPitch = img.GetRowPitch();
```

```
for (y = 0; y < height; y++)
{
    pixelPtr = rowPtr;

    for (x = 0; x < width; x++)
    {
        pixelValue = *pixelPtr;

        // Add your pixel computation code here

        *pixelPtr = pixelValue;
        pixelPtr++;
    }

    rowPtr += rowPitch;
}
```

ROI Placement

```
///////////////////////////////////////////////////////////////
// This code snippet shows how to attach an ROI to an image //
// and set its placement. //
///////////////////////////////////////////////////////////////
```

```
// Image constructor
EImageBW8 parentImage;
```

```
// ROI constructor
EROIBW8 myROI;
```

```
// ...
```

```
// Attach the ROI to the image
myROI.Attach(&parentImage);
```

```
//Set the ROI position
myROI.SetPlacement(50, 50, 200, 100);
```

Vector Management

```
///////////////////////////////////////////////////////////////
// This code snippet shows how to create a vector, fill it //
// and retrieve the value of a given element. //
///////////////////////////////////////////////////////////////
```

```
// EBW8Vector constructor
EBW8Vector ramp;
```

```
// Clear the vector
ramp.Empty();
```

```
// Fill the vector with increasing values
for(int i= 0; i < 128; i++)
{
        ramp.AddElement((EBW8)i);
}
```

```
// Retrieve the 10th element value
EBW8 value= ramp[9];
```

Exception Management

```
//////////////////////////////////////////
// This code snippet shows how to manage //
// Open eVision exceptions. //
//////////////////////////////////////////
```

```
try
{
        // Image constructor
        EImageC24 srcImage;
```

```
        // ...
```

```
        // Retrieve the pixel value at coordinates (56, 73)
        EC24 value= srcImage.GetPixel(56, 730);
}
```

```
catch(Euresys::Open_eVision_1_1::EException exc)
{
        // Retrieve the exception description
        std::string error = exc.What();
}
```

6.2. EasyImage

Thresholding

Single Thresholding

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform minimum residue //
// thresholding, absolute thresholding and relative //
// thresholding operations. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// ...
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Minimum residue thresholding (default method)
EasyImage::Threshold(&srcImage, &dstImage);
```

```
// Absolute thresholding (threshold = 110)
EasyImage::Threshold(&srcImage, &dstImage, 110);
```

```
// Relative thresholding (70% black, 30% white)
EasyImage::Threshold(&srcImage, &dstImage, EThresholdMode_Relative, 0, 255, 0.7f);
```

Double Thresholding

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform a thresholding //
// operation based on low and high threshold values. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// ...
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Double thresholding, low threshold = 50, high threshold = 150,
// pixels below 50 become black, pixels above 150 become white,
// pixels between thresholds become gray
EasyImage::DoubleThreshold(&srcImage, &dstImage, 50, 150, 0, 128, 255);
```

Histogram-Based Single Thresholding

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform a minimum residue //
// thresholding operation based on an histogram. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
```

```
EImageBW8 srcImage;
```

```
EImageBW8 dstImage;
```

```
// Histogram constructor
EBWHistogramVector histo;
```

```
// Variables
unsigned int thresholdValue;
float avgBelowThr, avgAboveThr;
```

```
// ...
```

```
// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);
```

```
// Compute the single threshold (and the average pixel values below and above the
threshold)
thresholdValue= EThresholdMode_MinResidue;
EasyImage::HistogramThreshold(&histo, thresholdValue, avgBelowThr, avgAboveThr);
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Perform the single thresholding
EasyImage::Threshold(&srcImage, &dstImage, thresholdValue);
```

Histogram-Based Double Thresholding

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform a double thresholding //
// operation. The low and high threshold values are computed //
// according to the minimum residue method based on an histogram. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// Histogram constructor
EBWHistogramVector histo;
```

```
// Variables
EBW8 lowThr;
EBW8 highThr;
float avgBelowThr, avgBetweenThr, avgAboveThr;
```

```
// ...
```

```
// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);
```

```
// Compute the low and high threshold values automatically
// (and the average pixel values below, between and above the threshold)
EasyImage::ThreeLevelsMinResidueThreshold(&histo, lowThr, highThr, avgBelowThr,
avgBetweenThr, avgAboveThr);
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Perform the double thresholding
EasyImage::DoubleThreshold(&srcImage, &dstImage, lowThr.Value, highThr.Value);
```

Arithmetic and Logic Operations

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous //
// arithmetic and logic operations to images. //
////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcGray0, srcGray1, dstGray;
EImageC24 srcColor, dstColor;
```

```
// ...
```

```
// All images must have the same size
dstGray.SetSize(&srcGray0);
// ...
```

```
// Subtract srcGray1 from srcGray0
EasyImage::Oper(EArithmeticLogicOperation_Subtract, &srcGray0, &srcGray1, &dstGray);
```

```
// Multiply srcGray0 by a constant value
EasyImage::Oper(EArithmeticLogicOperation_Multiply, &srcGray0, (EBW8)2, &dstGray);
```

```
// Add a constant value to srcColor
EasyImage::Oper(EArithmeticLogicOperation_Add, &srcColor, EC24(128,64,196),
&dstColor);
```

```
// Erase (blacken) the destination image where the source image is black
EasyImage::Oper(EArithmeticLogicOperation_SetZero, &srcGray0, (EBW8)0, &dstGray);
```

Convolution

Pre-Defined Kernel Filtering

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous //
// convolution operations based on pre-defined kernels. //
////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// ...
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Perform a Uniform filtering (5x5 kernel)
EasyImage::ConvolUniform(&srcImage, &dstImage, 2);

// Perform a Highpass filtering
EasyImage::ConvolHighpass1(&srcImage, &dstImage);

// Perform a Gradient filtering
EasyImage::ConvolGradient(&srcImage, &dstImage);

// Perform a Sobel filtering
EasyImage::ConvolSobel(&srcImage, &dstImage);
```

User-Defined Kernel Filtering

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply a convolution //
// operation based on a user-defined kernel. //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Create and define a user-defined kernel
// (Frei-Chen row gradient, positive only)
EKernel kernel;
kernel.SetKernelData(0.2929f, 0, -0.2929f,
 0.4142f, 0, -0.4142f,
 0.2929f, 0, -0.2929f);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Apply the convolution kernel
EasyImage::ConvolKernel(&srcImage, &dstImage, &kernel);
```

Non-Linear Filtering

Morphological Filtering

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous //
// morphological filtering operations. //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform an erosion (3x3 square kernel)
EasyImage::ErodeBox(&srcImage, &dstImage, 1);
```

```
// Perform a dilation (5x3 rectangular kernel)
EasyImage::DilateBox(&srcImage, &dstImage, 2, 1);
```

```
// Perform an Open operation (5x5 circular kernel)
EasyImage::OpenDisk(&srcImage, &dstImage, 2);
```

Hit-and-Miss Transform

```
////////////////////////////////////////////////////////////
// This code snippet shows how to highlight the left corner //
// of a rhombus by means of a Hit-and-Miss operation. //
////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;
```

```
// ...
```

```
// Create and define a Hit-and-Miss kernel
// corresponding to the left corner of a rhombus
EHitAndMissKernel leftCorner(-1, -1, 1, 1);
```

```
// Left column of the kernel
```

```
leftCorner.SetValue(-1, 0, EHitAndMissValue_Background);
```

```
// Middle column of the kernel
leftCorner.SetValue(0, -1, EHitAndMissValue_Background);
leftCorner.SetValue(0, 0, EHitAndMissValue_Foreground);
leftCorner.SetValue(0, 1, EHitAndMissValue_Background);
```

```
// Right column of the kernel
leftCorner.SetValue(1, -1, EHitAndMissValue_Foreground);
leftCorner.SetValue(1, 0, EHitAndMissValue_Foreground);
leftCorner.SetValue(1, 1, EHitAndMissValue_Foreground);
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Apply the Hit-and-Miss kernel
EasyImage::HitAndMiss(&srcImage, &dstImage, leftCorner);
```

Vector Operations

Path Sampling

```
////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve and store the //
// pixel values along a given path together with the //
// corresponding pixel coordinates. //
////////////////////////////////////////////////////////////
```

```
// Image constructor
EImageBW8 srcImage;
```

```
// ...
```

```
// Vector constructor
EBW8PathVector path;
```

```
// Path definition
path.Empty();
for (int i = 0; i < 100; i++)
{
```

```
 EBW8Path p;
 p.X = i;
 p.Y = i;
 p.Pixel = 128;
 path.AddElement(p);
}
```

```
// Get the image data along the path
EasyImage::ImageToPath(&srcImage, &path);
```

Profile Sampling

```
////////////////////////////////////////////////////////////
// This code snippet shows how to set, retrieve and store //
// the pixel values along a given line segment. //
////////////////////////////////////////////////////////////
```

```
// Image constructor
EImageBW8 srcImage;
```

```
// ...
```

```
// Vector constructor
EBW8Vector profile;
```

```
// Get the image data along segment (10,510)-(500,40)
EasyImage::ImageToLineSegment(&srcImage, &profile, 10, 510, 500, 40);
```

```
// Set all these points to white (255) in the image
EasyImage::LineSegmentToImage(&srcImage, 255, 10, 510, 500, 40);
```

Statistics

Image Statistics

```
////////////////////////////////////////////////////////////////////////
// This code snippet shows how to compute basic image statistics. //
////////////////////////////////////////////////////////////////////////
```

```
// Image constructor
EImageBW8 srcImage;
```

```
// ...
```

```
// Count the number of pixels above the threshold (128)
INT32 count;
EasyImage::Area(&srcImage, 128, count);
```

```
// Compute the pixels' average and standard deviation values
float stdDev, average;
EasyImage::PixelStdDev(&srcImage, stdDev, average);
```

```
// Compute the image gravity center (pixels above threshold)
float x, y;
EasyImage::GravityCenter(&srcImage, 128, x, y);
```

Sliding Windows Statistics

```
////////////////////////////////////////////////////////////////////////
// This code snippet shows how to perform sliding windows statistics. //
////////////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage0, dstImage1;
```

```
// ...
```

```
// All images must have the same size
dstImage0.SetSize(&srcImage);
dstImage1.SetSize(&srcImage);
```

```
// Local average in a 11x11 window
EasyImage::LocalAverage(&srcImage, &dstImage0, 5, 5);
```

```
// Local deviation in a 11x11 window
EasyImage::LocalDeviation(&srcImage, &dstImage1, 5, 5);
```

Histogram-Based Statistics

```
////////////////////////////////////////////////////////
// This code snippet shows how to compute statistics //
// based on an histogram. //
////////////////////////////////////////////////////////
```

```
// Image constructor
EImageBW8 srcImage;
```

```
// ...
```

```
// Histogram constructor
EBWHistogramVector histo;
```

```
// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);
```

```
// Compute the average gray-level value
float average = EasyImage::AnalyseHistogram(&histo, EHistogramFeature_
AveragePixelValue, 0, 255);
```

```
// Compute the gray-level standard deviation
float deviation = EasyImage::AnalyseHistogram(&histo, EHistogramFeature_
PixelValueStdDev, 0, 255);
```

Noise Reduction by Integration

Temporal Noise Reduction

```
////////////////////////////////////////////////
// This code snippet shows how to perform noise //
// reduction by temporal averaging. //
////////////////////////////////////////////////

// Images constructor
EImageBW16 noisyImage, cleanImage;

// 16 bits work image used as an accumulator
EImageBW16 store;

// ...

// All images must have the same size
cleanImage.SetSize(&noisyImage);
store.SetSize(&noisyImage);

// Clear the accumulator image
EasyImage::Oper(EArithmeticLogicOperation_Copy, (EBW16)0, &store);

// Accumulation loop
int n;
for (n=0; n < 10; n++)
{
 // Acquire a new image into noisyImage
 // ...

 // Add this new noisy image into the accumulator
 EasyImage::Oper(EArithmeticLogicOperation_Add, &noisyImage, &store, &store);
}

// Perform noise reduction
EasyImage::Oper(EArithmeticLogicOperation_Divide, &store, (EBW16)n, &cleanImage);
```

Recursive Average

```
////////////////////////////////////////////////
// This code snippet shows how to perform noise //
// reduction by recursive averaging. //
////////////////////////////////////////////////

// Images constructor
EImageBW8 noisyImage, cleanImage;

// 16 bits work image used as an accumulator
EImageBW16 store;

// ...

// All images must have the same size
cleanImage.SetSize(&noisyImage);
store.SetSize(&noisyImage);

// Clear the accumulator image
EasyImage::Oper(EArithmeticLogicOperation_Copy, (EBW16)0, &store);

// Prepare the transfer lookup table (reduction factor = 3)
EBW16Vector lut;
EasyImage::SetRecursiveAverageLUT(&lut, 3.f);
```

```
// Perform the noise reduction
EasyImage::RecursiveAverage(&noisyImage, &store, &cleanImage, &lut);
```

Feature Point Detectors

Harris Corner Detector

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve corners' coordinates //
// by means of the Harris corner detector algorithm. //
////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Harris corner detector
EHarrisCornerDetector harris;
EHarrisInterestPoints interestPoints;
harris.SetIntegrationScale(2.f);

// Perform the corner detection
harris.Apply(srcImage, interestPoints);

// Retrieve the number of corners
unsigned int index = interestPoints.GetPointCount();

// Retrieve the first corner coordinates
EPoint point = interestPoints.GetPoint(0);
float x = point.GetX();
float y = point.GetY();
```

Canny Edge Detector

```
////////////////////////////////////////////////////
// This code snippet shows how to highlight edges //
// by means of the Canny edge detector algorithm. //
////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Canny edge detector
ECannyEdgeDetector canny;

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the edges detection
canny.Apply(srcImage, dstImage);
```

Using Flexible Masks

Computing Pixels Average

```
////////////////////////////////////////////////////////
// This code snippet shows how to compute statistics //
// inside a region defined by a flexible mask. //
////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 srcImage;
EImageBW8 mask;
```

```
// ...
```

```
// Compute the average value of the source image pixels
// corresponding to the mask do-care areas only
float average;
EasyImage::PixelAverage(&srcImage, &mask, average);
```

6.3. EasyColor

Colorimetric Systems Conversion

```
////////////////////////////////////////////////////////
// This code snippet shows how to convert a color image //
// from the RGB to the Lab colorimetric system. //
////////////////////////////////////////////////////////
```

```
// Images constructor
EImageC24 srcImage;
EImageC24 dstImage;
```

```
// ...
```

```
// Prepare a lookup table for
// the RGB to La*b* conversion
EColorLookup lookup;
lookup.ConvertFromRgb(EColorSystem_Lab);
```

```
// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Perform the color conversion
EasyColor::Transform(&srcImage, &dstImage, &lookup);
```

Color Components

```
////////////////////////////////////////////////////////
// This code snippet shows how to create a color image //
// from 3 grayscale images and extract the luminance //
// component from a color image. //
////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 red, green, blue;
EImageC24 colorImage;
EImageBW8 luminance;
```

```
// ...
```

```
// Source and destination images must have the same size
colorImage.SetSize(&red);
```

```
// Combine the color planes into a color image
EasyColor::Compose(&red, &green, &blue, &colorImage);
```

```
// Prepare a lookup table for
// the RGB to LSH conversion
EColorLookup lookup;
lookup.ConvertFromRgb(EColorSystem_Lsh);
```

```
// Source and destination images must have the same size
luminance.SetSize(&colorImage);
```

```
// Get the Luminance component
EasyColor::GetComponent(&colorImage, &luminance, 0, &lookup);
```

White Balance

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform white balancing. //
////////////////////////////////////////////////////////////

// Images constructor
EImageC24 srcImage, dstImage;
EImageC24 whiteRef;

// ...

// Create a lookup table
EColorLookup lut;

// Measure the calibration values from a white reference image
float r, g, b;
EasyImage::PixelAverage(&whiteRef, r, g, b);

// Prepare the lookup table for
// a white balance operation
lut.WhiteBalance(1.00f, EasyColor::GetCompensateNtscGamma(), r, g, b);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the white balance operation
lut.Transform(&srcImage, &dstImage);
```

Pseudo-Coloring

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform pseudo-coloring. //
////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageC24 dstImage;

// ...

// Create a pseudo-color lookup table
EPseudoColorLookup pcLut;

// Define a shade of pure tints, from red to blue
pcLut.SetShading(EC24(255, 0, 0), EC24(0, 0, 255), EColorSystem_Ish);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Generate the pseudo-colored image
EasyColor::PseudoColor(&srcImage, &dstImage, &pcLut);
```

Bayer Pattern Decoding

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform Bayer pattern decoding. //
////////////////////////////////////////////////////////////////
```

```
// Images constructor
EImageBW8 bayerImage;
EImageC24 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&bayerImage);

// Convert to true color with simple interpolation, default parity assumed
EasyColor::BayerToC24(&bayerImage, &dstImage);
```