

HAN AEA - Embedded Vision & Machine Learning

EVD1

Development

basics

By Hugo Arends

EVD1 – File overview

-  evdk5
 -  evdk_images
 -  evdk_operators
 -  evdk_sheets
 -  evdk_workspace_apps
 -  evdk_workspace_targets

*Top level
folder structure*

EVD1 – File overview

- 📁 evdk5
 - 📁 evdk_images
 - 📁 evdk_operators
 - 📄 coding_and_compression.c
 - 📄 coding_and_compression.h
 - 📄 graphics_algorithms.c
 - 📄 graphics_algorithms.h
 - 📄 ...
 - 📁 evdk_sheets
 - 📁 evdk_workspace_apps
 - 📁 evdk_workspace_targets

*Image processing
source files,
a file per class*

EVD1 – File overview

- 📁 evdk5
 - 📁 evdk_images
 - 📁 evdk_operators
 - 📁 evdk_sheets
 - 📁 evdk_workspace_apps
 - 📁 evdk5_histogram_webcam
 - 📁 evdk5_img_from_file
 - 📁 evdk5_unit_test
 - 📁 evdk5_webcam
 - 📄 evdk5_apps.code-workspace
 - 📁 evdk_workspace_targets

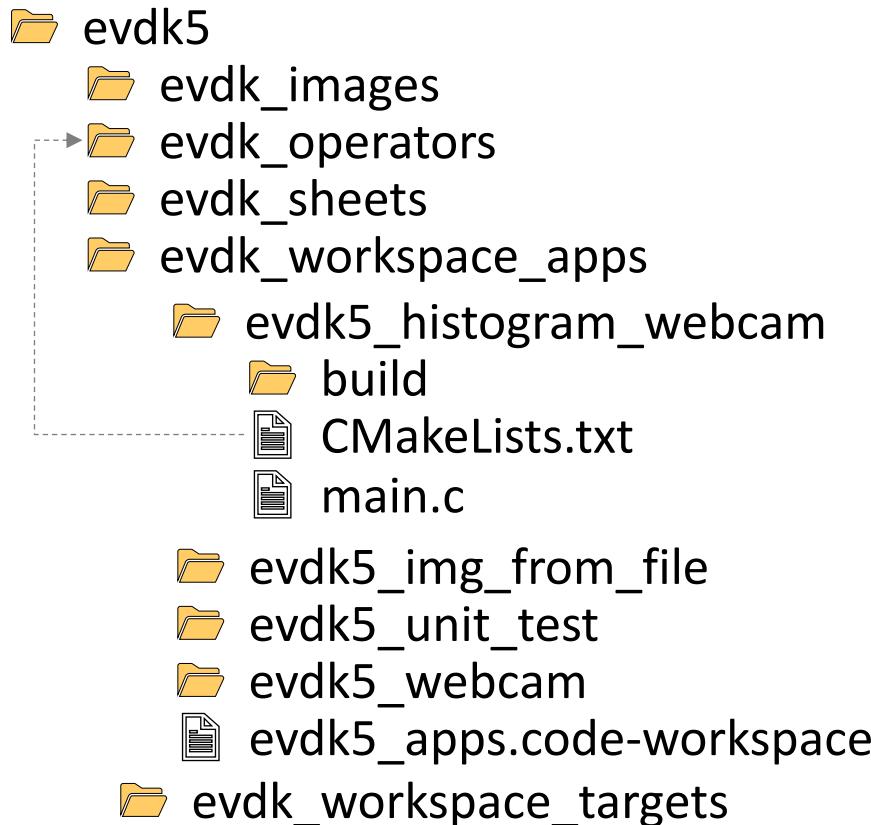
*Workspace for
PC apps*

EVD1 – File overview

- 📁 evdk5
 - 📁 evdk_images
 - 📁 evdk_operators
 - 📁 evdk_sheets
 - 📁 evdk_workspace_apps
 - 📁 evdk5_histogram_webcam
 - 📁 build
 - 📄 CMakeLists.txt
 - 📄 main.c
 - 📁 evdk5_img_from_file
 - 📁 evdk5_unit_test
 - 📁 evdk5_webcam
 - 📄 evdk5_apps.code-workspace
 - 📁 evdk_workspace_targets

*Workspace for
PC apps*

EVD1 – File overview



*Workspace for
PC apps*

EVD1 – File overview

- 📁 evdk5
 - 📁 evdk_images
 - 📁 evdk_operators
 - 📁 evdk_sheets
 - 📁 evdk_workspace_apps
 - 📁 evdk_workspace_targets
 - 📁 frdmmcxn947_evdk5_0
 - 📄 evdk5_targets.code-workspace

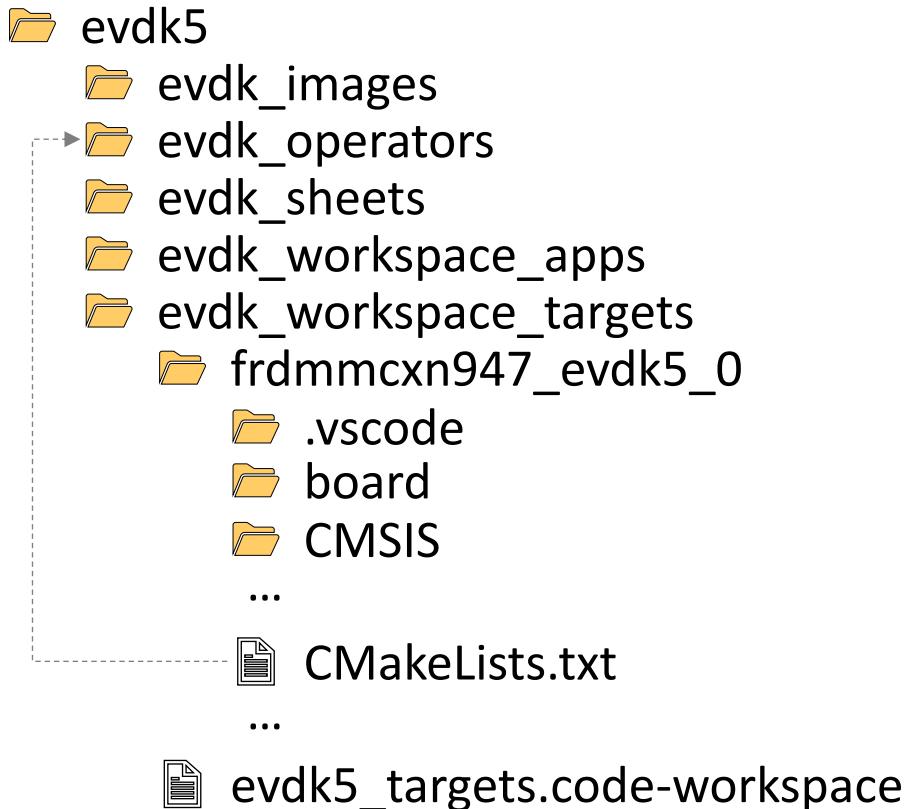
*Workspace for
target executables*

EVD1 – File overview

-  evdk5
 -  evdk_images
 -  evdk_operators
 -  evdk_sheets
 -  evdk_workspace_apps
 -  evdk_workspace_targets
 -  frdmmcxn947_evdk5_0
 -  .vscode
 -  board
 -  CMSIS
 - ...
-  CMakeLists.txt
- ...
-  evdk5_targets.code-workspace

*Workspace for
target executables*

EVD1 – File overview

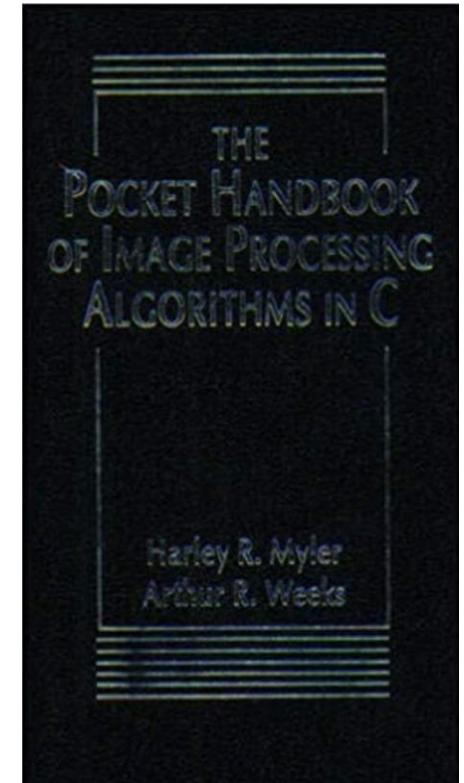


*Workspace for
target executables*

Image basics – attributes

```
/// Defines the attributes of an image
typedef struct
{
    int32_t      cols; ///< Number of columns in the image
    int32_t      rows; ///< Number of rows in the image
    eImageType    type; ///< The type of pixels in the image
    uint8_t     *data; ///< A pointer to the pixel data

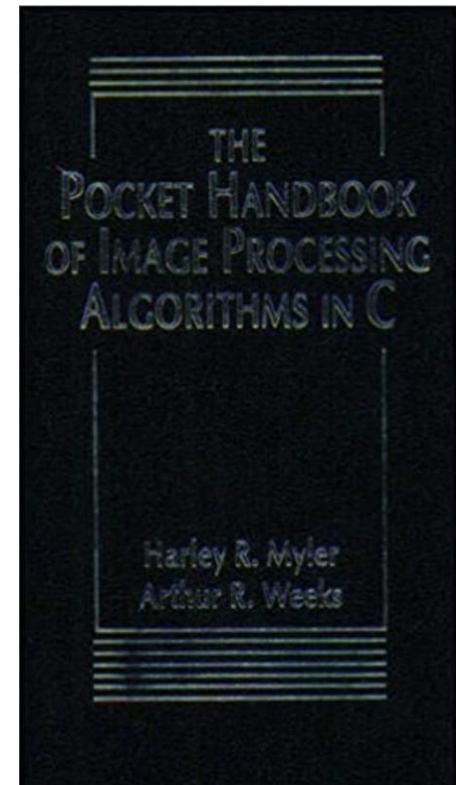
}image_t;
```



Myler, H. R., & Weeks, A. R. (2009). *The pocket handbook of image processing algorithms in C*. Prentice Hall Press.

Image basics – elmageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8  = 1, ///< Pixel type ::uint8_pixel_t.
    IMGTYPE_INT16  = 2, ///< Pixel type ::int16_pixel_t.
    IMGTYPE_INT32  = 4, ///< Pixel type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, ///< Pixel type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, ///< Pixel type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, ///< Pixel type ::bgr888_pixel_t.
} eImageType;
```



Myler, H. R., & Weeks, A. R. (2009). *The pocket handbook of image processing algorithms in C*. Prentice Hall Press.

Image basics – eImageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8   = 1, //Pixels of type ::uint8_pixel_t.
    IMGTYPE_INT16   = 2, //Pixels of type ::int16_pixel_t.
    IMGTYPE_INT32   = 4, //Pixels of type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, //Pixels of type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, //Pixels of type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, //Pixels of type ::bgr888_pixel_t.
}eImageType;
```

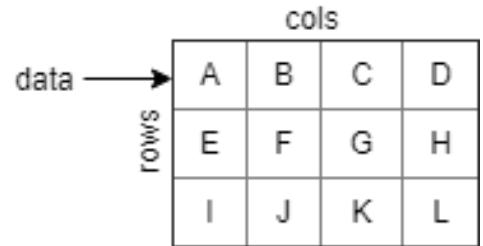
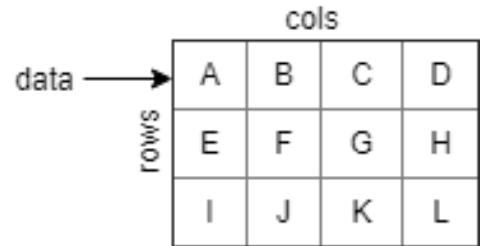


Image basics – elmageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8  = 1, ///< Pixels of type ::uint8_pixel_t.
    IMGTYPE_INT16  = 2, ///< Pixels of type ::int16_pixel_t.
    IMGTYPE_INT32  = 4, ///< Pixels of type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, ///< Pixels of type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, ///< Pixels of type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, ///< Pixels of type ::bgr888_pixel_t.
} eImageType;
```

```
/// \brief Type definition of a uint8 pixel
///
/// 8 bits per pixel
typedef uint8_t uint8_pixel_t;
```



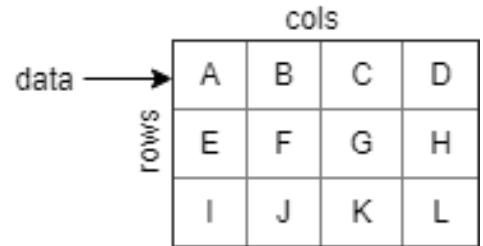
Memory allocation: $12 \times 1 \text{ byte} = 12 \text{ bytes}$



Image basics – eImageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8  = 1, ///< Pixels of type ::uint8_pixel_t.
    IMGTYPE_INT16  = 2, ///< Pixels of type ::int16_pixel_t.
    IMGTYPE_INT32  = 4, ///< Pixels of type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, ///< Pixels of type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, ///< Pixels of type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, ///< Pixels of type ::bgr888_pixel_t.
}eImageType;
```

```
/// \brief Type definition of an int16 pixel
///
/// 16 bits per pixel
typedef int16_t int16_pixel_t;
```



Memory allocation: $12 \times 2 \text{ bytes} = 24 \text{ bytes}$

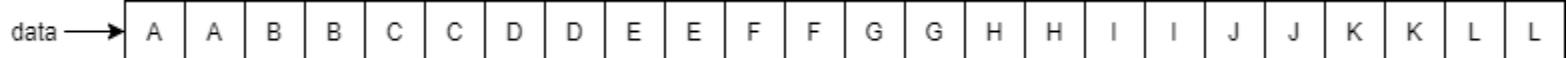
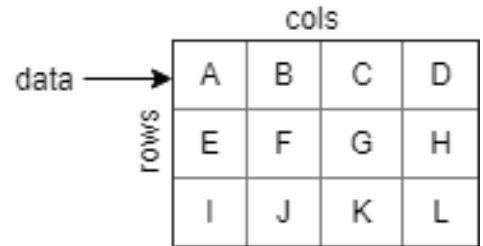


Image basics – eImageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8  = 1, ///< Pixels of type ::uint8_pixel_t.
    IMGTYPE_INT16  = 2, ///< Pixels of type ::int16_pixel_t.
    IMGTYPE_INT32  = 4, ///< Pixels of type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, ///< Pixels of type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, ///< Pixels of type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, ///< Pixels of type ::bgr888_pixel_t.
}eImageType;
```

```
/// \brief Type definition of an int32 pixel
///
/// 32 bits per pixel
typedef int32_t int32_pixel_t;
```



Memory allocation: $12 \times 4 \text{ bytes} = 48 \text{ bytes}$

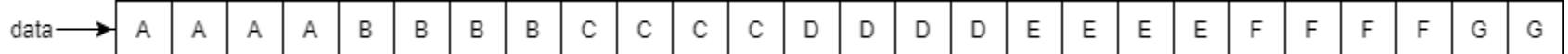
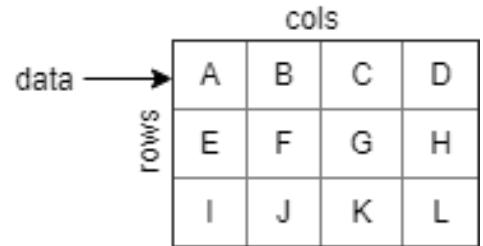


Image basics – elmageType

```
/// Defines the type of images
typedef enum
{
    IMGTYPE_UINT8  = 1, //Pixels of type ::uint8_pixel_t.
    IMGTYPE_INT16  = 2, //Pixels of type ::int16_pixel_t.
    IMGTYPE_INT32  = 4, //Pixels of type ::int32_pixel_t.
    IMGTYPE_FLOAT   = 8, //Pixels of type ::float_pixel_t.
    IMGTYPE_UYVY   = 16, //Pixels of type ::uyvy_pixel_t.
    IMGTYPE_BGR888 = 32, //Pixels of type ::bgr888_pixel_t.
}eImageType;
```

```
/// \brief Type definition of a float pixel
///
/// 32 bits per pixel
typedef float float_pixel_t;
```



Memory allocation: $12 \times 4 \text{ bytes} = 48 \text{ bytes}$

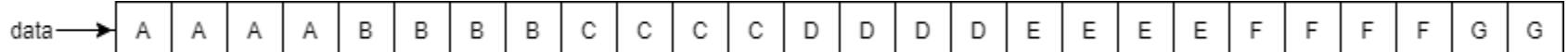


Image basics – Creating images

```
// Create an image
image_t *src = newUint8Image(4, 3);

// Use src in an image processing pipeline
// ...

// Cleanup
deleteUint8Image(src);
```

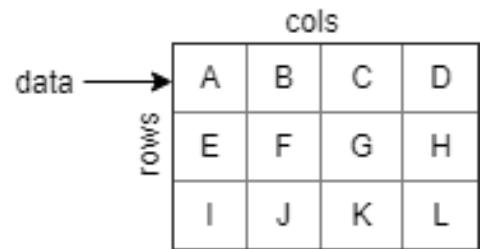


Image basics – Creating images

```
// Create an image
image_t *src = newFloatImage(4, 3);

// Use src in an image processing pipeline
// ...

// Cleanup
deleteFloatImage(src);
```

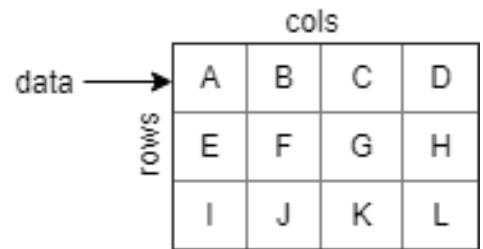
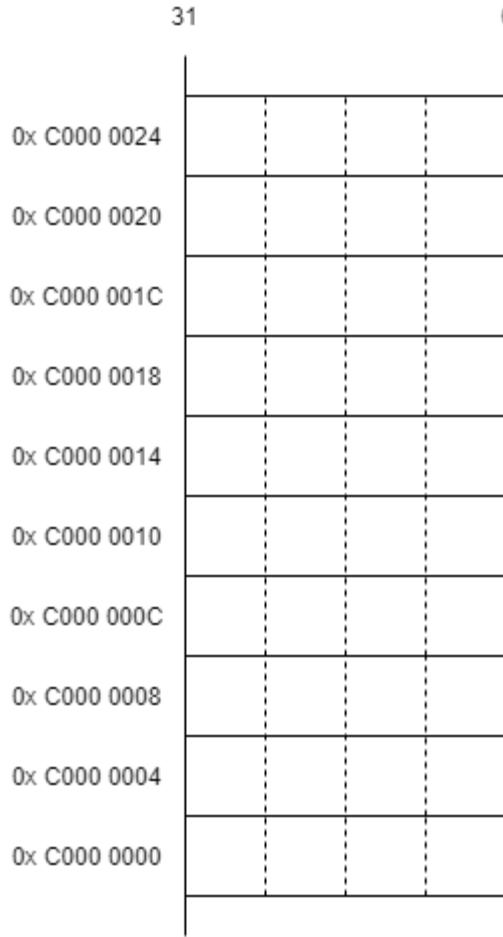


Image basics – Pointers



```
int a = 0;
```

- ‘a’ is a variable, but what is a variable?
name of a storage area
- What does the type of a variable tell?
size and layout in memory
- How can we get the memory address of a variable?
by using the reference operator: &
- Why is this incorrect?

```
char *p2 = &a;
```

The base-type of the pointer is different from the base-type of the variable

Image basics – Pointers

	31		0
0x C000 0024			
0x C000 0020			
0x C000 001C			
0x C000 0018		0x0000	
0x C000 0014	0xC0000000		
0x C000 0010			0x00
0x C000 000C	0xC0000000		
0x C000 0008		0x09	0x08
0x C000 0004	0x07	0x06	0x05
0x C000 0000	0x03	0x02	0x01

```
uint8_t data[10] = {0,1,2,3,4,5,6,7,8,9};  
uint8_t *p = data; // alternative: &data[0]  
uint8_t a = 0;  
uint16_t *q = (uint16_t *)data;  
uint16_t b = 0;
```

```
// Reading one element from the data array  
a = data[3]; // a = 0x03  
a = *(data+3); // a = 0x03  
a = *(p+3); // a = 0x03  
  
// Reading two elements from the data array  
b = *(q+3); // b = 0x0706
```

Image basics – Pointers

	31		0
0x C000 0024			
0x C000 0020			
0x C000 001C			
0x C000 0018		0x0000	
0x C000 0014	0xC0000000		
0x C000 0010			0x00
0x C000 000C	0xC0000000		
0x C000 0008		0x09	0x08
0x C000 0004	0x07	0x06	0x05
0x C000 0000	0x03	0x02	0x01

```
uint8_t data[10] = {0,1,2,3,4,5,6,7,8,9};  
uint8_t *p = data; // alternative: &data[0]  
uint8_t a = 0;  
uint16_t *q = (uint16_t *)data;  
uint16_t b = 0;
```

```
// Writing one element to the data array  
p = data + 1; // p = 0xC000 0001  
p++; // p = 0xC000 0002  
*p = 0; // data = {0,1,0,3,4,5,6,7,8,9}  
  
// Writing two elements to the data array  
q = (uint16_t *)data + 1; // q = 0xC000 0002  
q++; // q = 0xC000 0004  
*q = 0; // data = {0,1,2,3,0,0,6,7,8,9}
```

Image basics – Pointers

	31		0
0x C000 0024			
0x C000 0020			
0x C000 001C			
0x C000 0018		0x0000	
0x C000 0014	0xC0000000		
0x C000 0010			0x00
0x C000 000C	0xC0000000		
0x C000 0008		0x09	0x08
0x C000 0004	0x07	0x06	0x05
0x C000 0000	0x03	0x02	0x01

```
uint8_t data[10] = {0,1,2,3,4,5,6,7,8,9};  
uint8_t *p = data; // alternative: &data[0]  
uint8_t a = 0;  
uint16_t *q = (uint16_t *)data;  
uint16_t b = 0;
```

```
// Be careful with typecasting!  
p = data + 1; // p = 0xC000 0001  
q = (uint16_t *)data + 1; // q = 0xC000 0002  
q = (uint16_t *)(data + 1); // q = 0xC000 0001
```

Image basics – Pointers

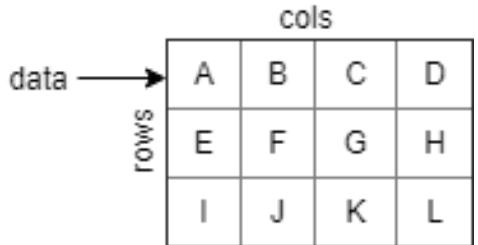
	31		0
0x C000 0024			
0x C000 0020			
0x C000 001C	0xC000 000C		
0x C000 0018	0xC000 0000		
0x C000 0014	0x0000 0000		
0x C000 0010	0x0000 0002		
0x C000 000C	0x0000 0000		
0x C000 0008		0x09	0x08
0x C000 0004	0x07	0x06	0x05
0x C000 0000	0x03	0x02	0x01

```
uint8_t data[10] = {0,1,2,3,4,5,6,7,8,9};  
  
image_t image = {0,2,IMGTYPE_UINT8,data};  
  
image_t *src = &image;
```

```
// Image manipulation  
image.cols = 5; // image = {5,2,0,0xC000 0000}  
src->cols = 5; // image = {5,2,0,0xC000 0000}  
  
*((uint8_t *)src->data + 3) = 0; // data={0,1,2,0,4,5,6,7,8,9}  
  
*((uint16_t *)src->data + 3) = 0; // data={0,1,2,3,4,5,0,0,8,9}
```

Image basics – Accessing pixels

Use convenience functions for accessing pixels



```
inline void setUint8Pixel(const image_t *img, const int32_t c, const int32_t r, const uint8_pixel_t value)
{
    *((uint8_pixel_t *) (img->data) + (r * img->cols + c)) = value;
}
```

Explicit type cast to
pixel type

Calculating
the offset

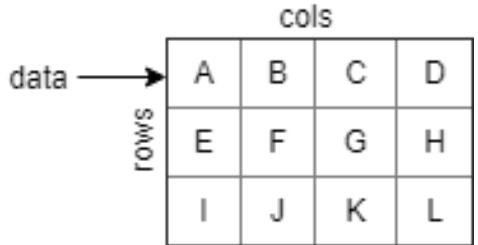
```
inline uint8_pixel_t getUint8Pixel(const image_t *img, const int32_t c, const int32_t r)
{
    return (*((uint8_pixel_t *) (img->data) + (r * img->cols + c)));
}
```

Explicit type cast to
pixel type

Calculating
the offset

Image basics – Accessing pixels

Use convenience functions for accessing pixels



```
inline void setFloatPixel(const image_t *img, const int32_t c, const int32_t r, const float_pixel_t value)
{
    *((float_pixel_t *) (img->data) + (r * img->cols + c)) = value;
}
```

Explicit type cast to
pixel type

Calculating
the offset

```
inline float_pixel_t getFloatPixel(const image_t *img, const int32_t c, const int32_t r)
{
    return (*((float_pixel_t *) (img->data) + (r * img->cols + c)));
}
```

Explicit type cast to
pixel type

Calculating
the offset

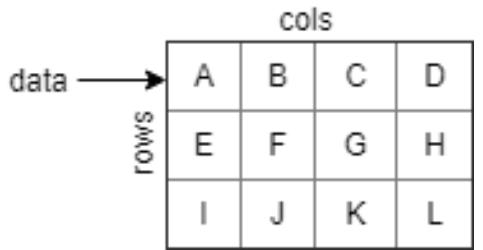
Image basics – Accessing pixels

```
// Create a new image
image_t *src = newUint8Image(4, 3);

// Clear the image
clearImage(src);

// Get the value of pixel B (1,0)
if(getUint8Pixel(src, 1, 0) == 0)
{
    // Set pixel G (2,1) to the value 100
    setUint8Pixel(src, 2, 1, 100);
}

// Cleanup
deleteUint8Image(src);
```



	cols			
rows	A	B	C	D
E	F	G	H	
I	J	K	L	

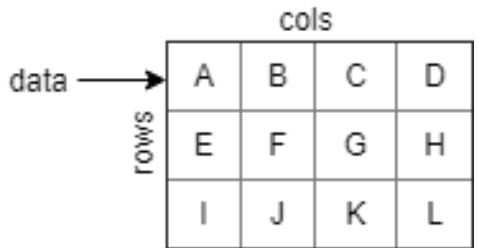
Image basics – Accessing pixels

```
// Create a new image
image_t *src = newFloatImage(4, 3);

// Clear the image
clearImage(src);

// Get the value of pixel B (1,0)
if(getFloatPixel(src, 1, 0) == 0)
{
    // Set pixel G (2,1) to the value 100
    setFloatPixel(src, 2, 1, 100);
}

// Cleanup
deleteFloatImage(src);
```



	cols			
rows	A	B	C	D
E	F	G	H	
I	J	K	L	

Anatomy of a project

Target projects

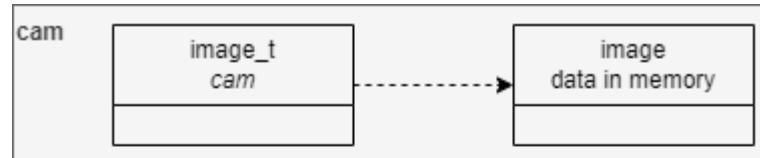
- *For running the image processing pipeline on the microcontroller*

Apps

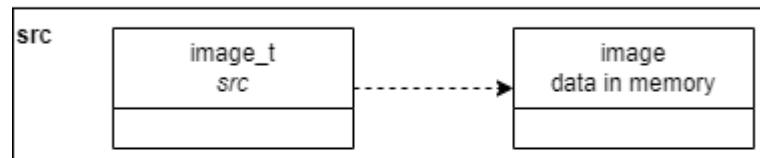
- *For running the image processing pipeline on the PC (uses OpenCV)*
- *For visualizing images from a target and displaying additional information, e.g. a histogram (uses OpenCV)*
- *For unit testing the individual image processing operators (uses Unity)*

Anatomy of a target project

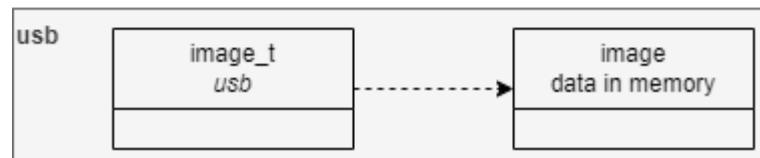
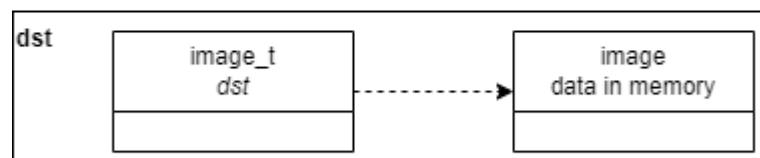
Mandatory



Optional



Mandatory

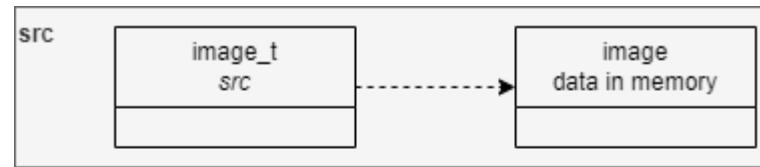


Anatomy of a target project

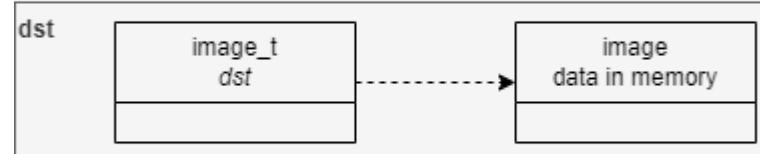
Demo

Anatomy of the unit test app project

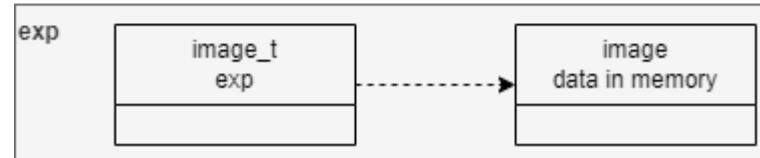
Mandatory



Mandatory



Mandatory

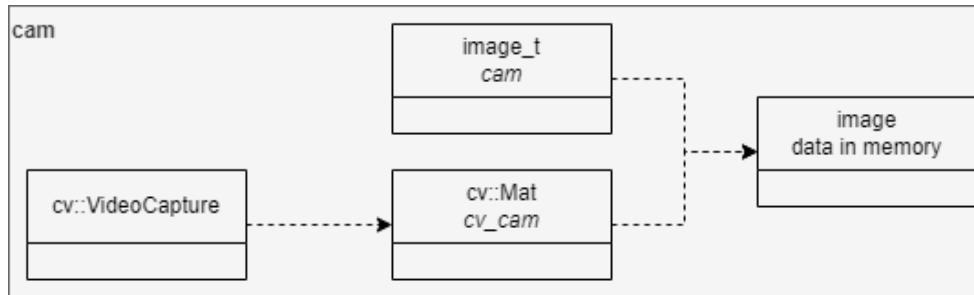


Anatomy of the unit test app project

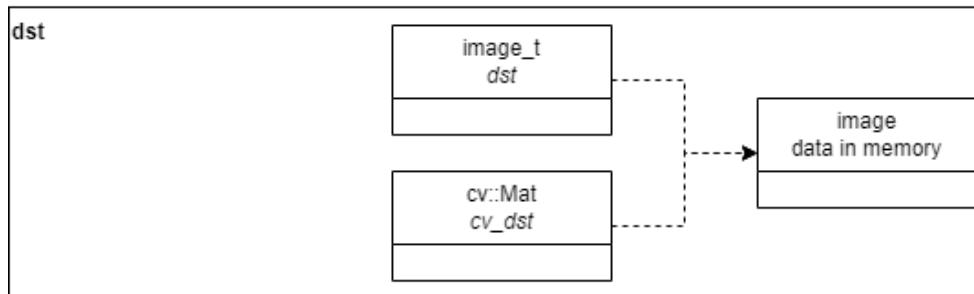
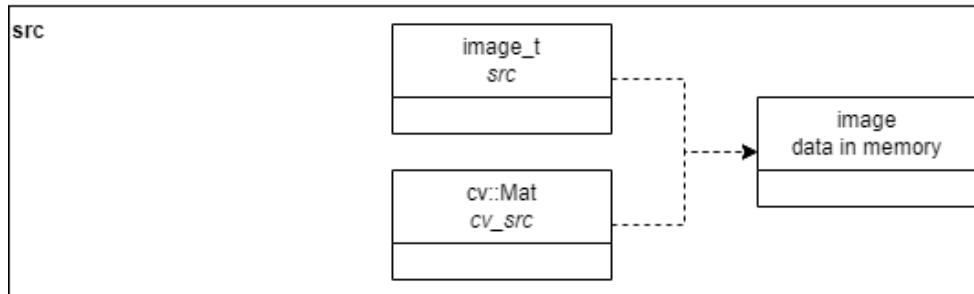
Demo

Anatomy of an app project

Mandatory



Optional



EVD1 – Assignment



Study guide **Week 1**

- 4 Change and run the example project
- 5 Run OpenCV Webcam app
- 6 Unit testing