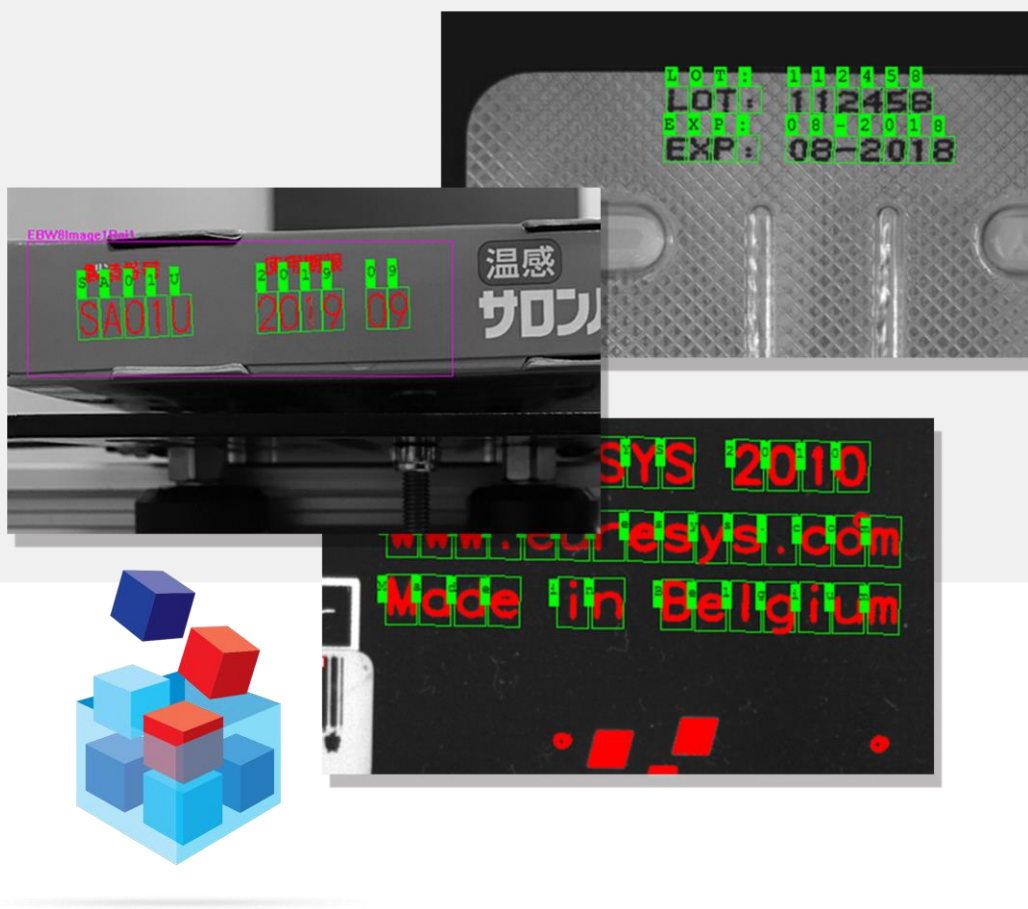


EasyOCR2



EURESYS s.a. shall retain all property rights, title and interest of the documentation of the hardware and the software, and of the trademarks of EURESYS s.a. All the names of companies and products mentioned in the documentation may be the trademarks of their respective owners. The licensing, use, leasing, loaning, translation, reproduction, copying or modification of the hardware or the software, brands or documentation of EURESYS s.a. contained in this book, is not allowed without prior notice. EURESYS s.a. may modify the product specification or change the information given in this documentation at any time, at its discretion, and without prior notice. EURESYS s.a. shall not be liable for any loss of or damage to revenues, profits, goodwill, data, information systems or other special, incidental, indirect, consequential or punitive damages of any kind arising in connection with the use of the hardware or the software of EURESYS s.a. or resulting of omissions or errors in this documentation.

Contents

Use Cases	4
Basic Usage	4
Using only specific types of reference characters.....	5
Using Reference Characters from Multiple Source Files.....	6
Retrieving In-Depth Text Information	7
Creating a Simple OCR2 Character Database by Learning	8
Improving an OCR2 Character Database by learning	9
Combining Character Databases from Different Sources.....	10
Advanced Parameters	11
Text Angle Restriction	11
Character Width Tolerance & Bias	11
Character Spacing Bias	11
Detection Delta and Max Variation	11
Maximum Character Fragmentation	11

Use Cases

Basic Usage

1. Create an **EOCR2** instance.
2. Set the expected size of the characters (mandatory).
3. Set the expected topology (mandatory).
4. Add reference characters to the character database (mandatory).
5. Optionally, set the text polarity (by default it is light characters on a dark background)
6. Read the text on a BW8 Image or ROI.

```
// Create an EOCR2 instance
EOCR2 instance;

// Set the expected character size
instance.SetCharsHeight(20);
instance.SetCharsWidth(15);

// Set the topology
instance.SetTopology("\.{4} [LN]{3} \n Lu{2}SN{4}");

// Add reference characters from file to the character database
instance.AddCharactersToDatabase(fileName);

// Set the text polarity
instance.SetTextPolarity(EasyOCR2TextPolarity_BlackOnWhite);

// Read text
std::string result = instance.Read(roi);
```

Note: The reference characters that are added to the database can be loaded either from a standard TrueType (.TTF/.TTC) font or from a character database created by EasyOCR2 (see further). You can load reference characters from more than one source, and mix character from TrueType files and OCR2 character databases.

Using only specific types of reference characters

1. Create an **EOCR2** instance.
2. Set the expected size of the characters (mandatory).
3. Set the expected topology (mandatory).
4. Add reference characters to the character database with a character filter (mandatory).
5. Optionally, set the text polarity (by default it is light characters on dark background)
6. Read the text on a BW8 Image or ROI.

```
// Create an EOCR2 instance
EOCR2 instance;

// Set the expected character size
instance.SetCharsHeight(20);
instance.SetCharsWidth(15);

// Set the topology
instance.SetTopology("\.{4} [LN]{3} \n Lu{2}SN{4}");

// Add reference characters from file to the character database, only digits
instance.AddCharactersToDatabase(fileName, EasyOCR2CharacterFilter_Digits);

// Set the text polarity
instance.SetTextPolarity(EasyOCR2TextPolarity_BlackOnWhite);

// Read text
std::string result = instance.Read(roi);
```

Note: The reference characters can be loaded either from a standard TrueType (.TTF/.TTC) font or from a character database created by EasyOCR2 (see further).

Using Reference Characters from Multiple Source Files

7. Create an **EOCR2** instance.
8. Set the expected size of the characters (mandatory).
9. Set the expected topology (mandatory).
10. Add reference characters from the first file, optionally with a filter (mandatory).
11. Add reference characters from the second file, optionally with a filter (optional).
12. Optionally, set the text polarity (by default it is light characters on dark background)
13. Read the text on a BW8 Image or ROI.

```
// Create an EOCR2 instance
EOCR2 instance;

// Set the expected character size
instance.SetCharsHeight(20);
instance.SetCharsWidth(15);

// Set the topology
instance.SetTopology(".{4} [LN]{3} \n Lu{2}SN{4}");

// Add reference characters from file to the character database
instance.AddCharactersToDatabase(fileName1, EasyOCR2CharacterFilter_Digits);
instance.AddCharactersToDatabase(fileName2, EasyOCR2CharacterFilter_Letters);

// Set the text polarity
instance.SetTextPolarity(EasyOCR2TextPolarity_BlackOnWhite);

// Read text
std::string result = instance.Read(roi);
```

Note: The reference characters can be loaded either from a standard TrueType (.TTF/.TTC) font or from a character database created by EasyOCR2 (see further).

Retrieving In-Depth Text Information

After the steps described in 1.1 or 1.2, you can retrieve more information on what was read using **EOCR2::GetReadText()**.

GetReadText() returns an **EOCR2Text** class instance, which contains, in addition to the read text, the position of the characters in the image and the full list of scored recognition candidates for each character.

```
// Retrieve the extended text information
EOCR2Text text = instance.GetReadText();

// Insert the reference text into the 'text' structure
std::vector<EOCR2Line> lines = text.GetLines();
for (int l = 0; l < lines.size(); l++)
{
    std::vector<EOCR2Word> words = lines[l].GetWords();
    for (int w = 0; w < words.size(); w++)
    {
        // Retrieve the bounding box of the current word
        ERectangle wordBox = words[w].GetBoundingBox();

        std::vector<EOCR2Char> chars = words[w].GetCharacters();
        for (int c = 0; c < chars.size(); c++)
        {
            // Retrieve all recognition candidates for the current char with their respective score
            std::vector<EOCR2CharacterCandidates> candidates = chars[c].GetCandidates();
        }
    }
}
```

Creating a Simple OCR2 Character Database by Learning

1. Create an **EOCR2** instance
2. Set the expected size of the characters (mandatory).
3. Set the expected topology (mandatory).
4. Optionally, set the text polarity (by default it is light characters on a dark background)
5. Call **EOCR2::Detect()**. Detect will return an **EOCR2Text** class instance.
6. Complete the **EOCR2Text** instance with text recognition information.
7. Call **EOCR2::Learn()** with the filled **EOCR2Text** class. This will add the detected characters with the provided information to the reference character database.
8. Save the database using **EOCR2::SaveCharacterDatabase()**.

```
// Create an EOCR2 instance
EOCR2 instance;

// Set the expected character size
instance.SetCharsHeight(20);
instance.SetCharsWidth(15);

// Set the topology
instance.SetTopology("\{4} [LN]{3} \n Lu{2}SN{4}");

// Set the text polarity
instance.SetTextPolarity(EasyOCR2TextPolarity_BlackOnWhite);

// Detect text
EOCR2Text text = instance.Detect(roi);

// Set reference string into text
text.SetText(referenceString);

// Learn reference text
text.Learn(text);

// Save the database
instance.SaveCharacterDatabase(fileName);
```

Note: You can call the **Detect()** then **Learn()** sequence multiple times to build a more complete font.

Note: You can also learn only part of the text by passing a component of the **EOCR2text** (**EOCR2Line**, **EOCR2Word**, **EOCR2Char**) instead of the whole **EOCR2text** to **Learn()**.

Improving an OCR2 Character Database by learning

1. Create an **EOCR2** instance
2. Set the expected size of the characters (mandatory).
3. Set the expected topology (mandatory).
4. Optionally, set the text polarity (by default it is light characters on a dark background)
5. Add reference characters from file to the character database. This will be the original character database.
6. Call **EOCR2::Detect()**. Detect will return an **EOCR2Text** class instance.
7. Call **EOCR2::Recognize()** on the returned **EOCR2Text** instance. This will fill the instance with the best recognition guess for the characters.
8. Correct errors in the recognition information found in the **EOCR2Text** instance.
9. Call **EOCR2::Learn()** with the corrected **EOCR2Text** class. This will improve the original character database.
10. Save the improved character database using **EOCR2::SaveCharacterDatabase()**.

```
// Create an EOCR2 instance
EOCR2 instance;

// Set the expected character size
instance.SetCharsHeight(20);
instance.SetCharsWidth(15);

// Set the topology
instance.SetTopology("\.{4} [LN]{3} \n Lu{2}SN{4}");

// Set the text polarity
instance.SetTextPolarity(EasyOCR2TextPolarity_BlackOnWhite);

// Add reference characters from file to the character database
instance.AddCharactersToDatabase(fileName);

// Detect text
EOCR2Text text = instance.Detect(roi);

// Recognize text
std::string result = instance.Recognize(text);

// Update character database if needed
if (result != referenceString)
{
    // Set reference string into text
    text.SetText(referenceString);

    // Learn reference text
    text.Learn(text);

    // Save the database
    instance.SaveCharacterDatabase(newFileName);
}
```

Note: You can also learn only part of the text by passing a component of the **EOCR2text** (**EOCR2Line**, **EOCR2Word**, **EOCR2Char**) instead of the whole **EOCR2text** to **Learn()**.

Note: The original character database can be generated either from a previously saved OCR2 database or from a TrueType font. The improved character database that is saved, however, will always be an OCR2 font.

Combining Character Databases from Different Sources

1. Create an **EOCR2** instance.
2. Add reference characters from the first source to the character database, optionally with a filter.
3. Add reference characters from the second source to the character database, optionally with a filter.
4. Save the combined font using **EOCR2 : : SaveCharacterDatabase ()**.

```
// Create an EOCR2 instance
EOCR2 instance;

// Add reference characters from the first file to the character database
instance.AddCharactersToDatabase(firstFileName);

// Add reference characters from the second file to the character database
instance.AddCharactersToDatabase(secondFileName, EasyOCR2CharacterFilter_Digits);

// Save the character database
instance.SaveCharacterDatabase(combinedFileName);
```

Note: You can combine characters from TrueType fonts and/or OCR2 character databases. What will be saved, however, will always be an OCR2 character database.

Advanced Parameters

Text Angle Restriction

The base angle parameters allow you to set the expected angle of the text with respect to the horizontal. The angle tolerance parameter allows you to restrict the range of angles that the text may have with respect to the base angle. In most cases, this will speed up the detection process.

The accessors used to set these restrictions are **EOCR2::Get/SetTextBaseAngle()** and **EOCR2::Get/SetTextAngleTolerance()**.

Character Width Tolerance & Bias

The **EOCR2::Set/GetCharsWidthTolerance()** parameter allows you to define a tolerance on the width of the characters, the detection algorithm will optimize the width of the bounding boxes within the limits defined by the tolerance. This parameter is expressed as a fraction of the character size.

The **EOCR2::Set/GetCharsWidthBias()** parameter allows you to define a bias toward narrower or wider bounding boxes during the optimization. Choices for the parameter are:

EasyOCR2CharWidthBias_Widest, **EasyOCR2CharWidthBias_Wide**,
EasyOCR2CharWidthBias_Neutral, **EasyOCR2CharWidthBias_Narrow** and
EasyOCR2CharWidthBias_Narrowest.

Character Spacing Bias

The **EOCR2::Set/GetCharsSpacingBias()** parameter allows you to define a bias toward narrower or wider spacing between the character bounding boxes during the optimization. Choices for the parameter are:

EasyOCR2CharSpacingBias_Wide, **EasyOCR2CharSpacingBias_Neutral** and
EasyOCR2CharSpacingBias_Narrow.

Detection Delta and Max Variation

Detection Delta and Max Variation are parameters pertaining to the segmentation phase of the algorithm. They should be used only if the results of the segmentation do not allow a reliable detection. The accessors to these parameters are **EOCR2::Get/SetDetectionDelta()** and **EOCR2::Get/SetMaxVariation()**.

Maximum Character Fragmentation

Character Fragmentation happens when a character is separated in pieces, like in a dot-matrix font or in a damaged character. The Maximum Character Fragmentation is defined as the size of the smallest piece of character that should be considered.

The accessors **EOCR2::Set/GetCharsMaxFragmentation()** are provided to set the parameter. It is expressed as a fraction of the character area (**CharsWidth * CharsHeight**).