Repairing corrupted JPEG images with *JPEG visual repair tool*

by Alberto Maccioni

This tutorial describes how to correct JPEG images that exhibit various data corruption artifacts; a short introduction to JPEG compression and corruption effects is available in chapters 3 and 4.

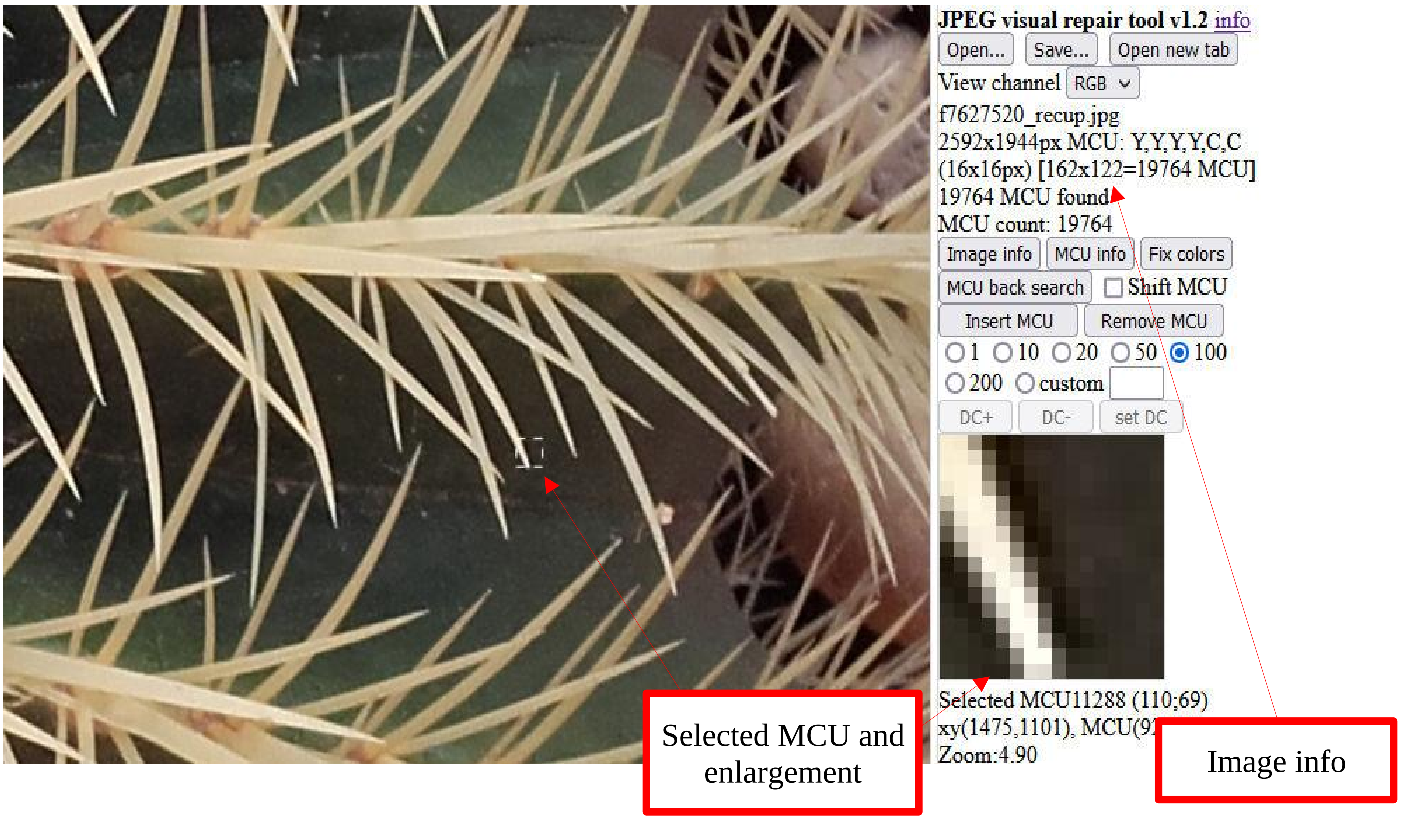
JPEG visual repair tool can load JPEG images while preserving MCU (Minimum Coded Unit) coding data and allows editing at MCU level.

It is written in JavaScript and resides in a single html document; it can be simply saved for offline use, there are no library dependencies. It is based on jpeg-decomp.

Using it you can:

- view image info
- delete, insert, copy, paste MCUs
- change DC level of each MCU
- view the image as RGB, Y, Cb, Cr
- automatically fix color differences
- view MCU pixel levels, coefficients, and binary data-stream

Here the main window after loading an image:



View channel RGB
f7627520_recup.jpg
2592x1944px MCU: Y,Y,Y,Y,C,C
(16x16px) [162x122=19764 MCU]
19764 MCU found
MCU count: 19764
☐ Shift MCU
○1 ○10 ○20 ○50 [✓ 100]
○200 ○custom
Selected MCU11288 (110;69)
xy(1475,1101), MCU(9
Zoom:4.90

Controls:

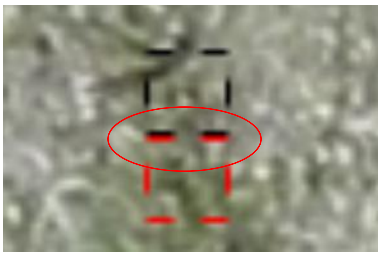
- **mouse wheel** → zoom image
- **left click** → select MCU; a black or white rectangle appears on selected MCU
- **shift+left click** → extend selection
- **left button drag** → drag image
- **right click** → select MCU for color fixing; a red rectangle appears on selected MCU
- **right button drag** → extend selection for color fixing
- **ctrl-c** → copy selected MCUs
- **ctrl-v** → paste before selected MCU
- **ctrl-z** → undo last operation
- **arrows** → change selected MCU
- **del** → delete MCU
- **i** → open MCU info dialog
- **s** → shift MCU rendering using left/right keys
- **1-2-3-4** → change view (RGB, Y, Cb, Cr)

A crossed red rectangle appears on MCUs that produced decode errors.

Fix colors

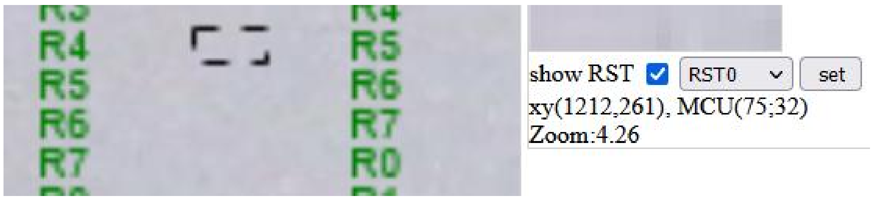
This function changes the DC coefficients in the selected MCU (for Y, Cb, Cr sub-blocks) in order to minimize the color difference with respect to the corresponding MCU in the previous line.

Top and bottom row of pixels are considered; for example, in the picture below, the minimization area is circled in red:



Restart markers

Additional functions are enabled after loading images with restart markers; it is possible to show the marker number over the image and set the RST marker value to any number:



show RST ✓ RST0 set
xy(1212,261), MCU(75;32)
Zoom:4.26

MCU back search

Experimental function: starting from the selected MCU, scans backwards to find all possible MCU combinations that end exactly at the selected one; the desired combination can be inserted in the image.

Limitations

The current version lacks support for the following JPEG features:

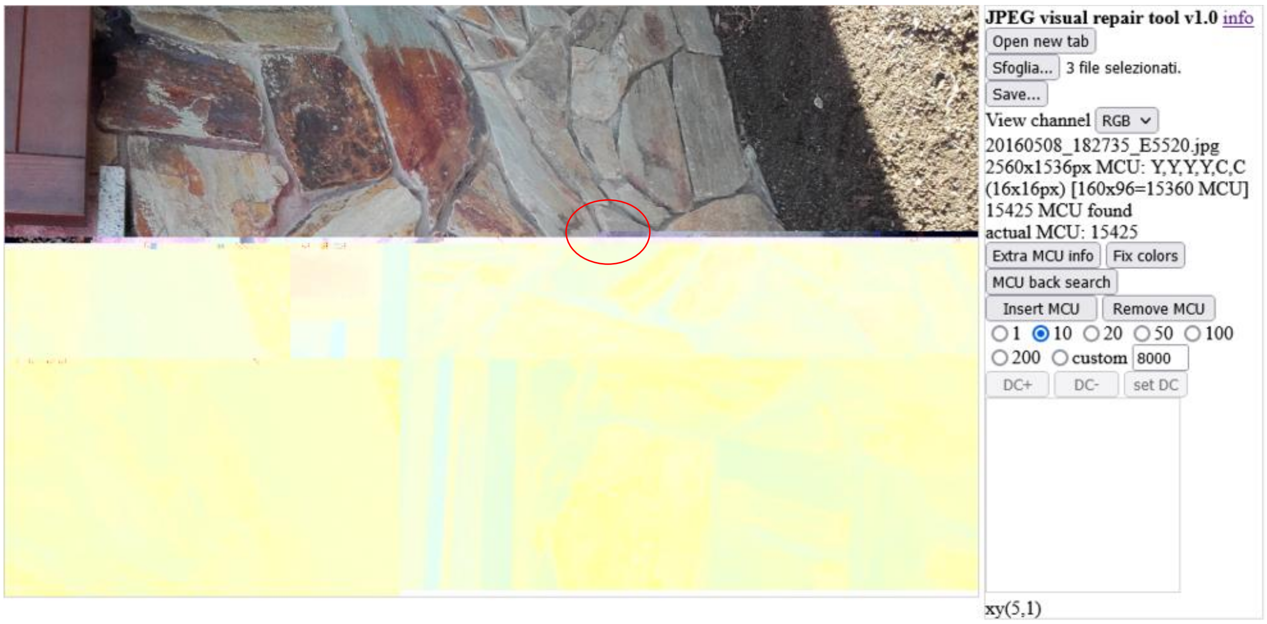
- progressive scan
- arithmetic encoding

Example of image repair

An example of repair flow is described here, applied to the following image:

This is how it appears to the file explorer and most image viewers.

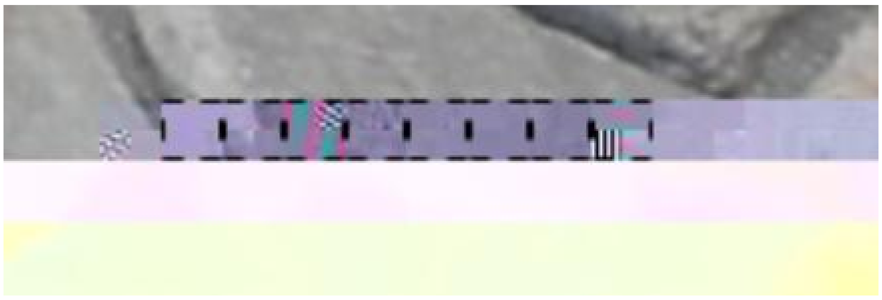
Let’s load it into JPEG visual repair tool:


JPEG visual repair tool v1.0 info
Open new tab
Sfoglia... 3 file selezionati.
Save...
View channel RGB
20160508_182735_E5520.jpg
2560x1536px MCU: Y,Y,Y,Y,C,C
(16x16px) [160x96=15360 MCU]
15425 MCU found
actual MCU: 15425

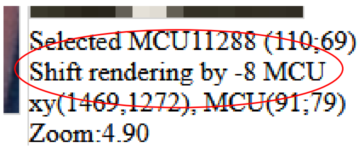
xy(5,1)

Notice how corruption starts in the circled area; other corruption points can be seen further down, but DC level corruption makes it hard to exactly locate them. 15425 MCUs were found instead of the original value of 15360; extra ones are a consequence of data corruption and contain wrong information.

Now zoom-in around the first point:

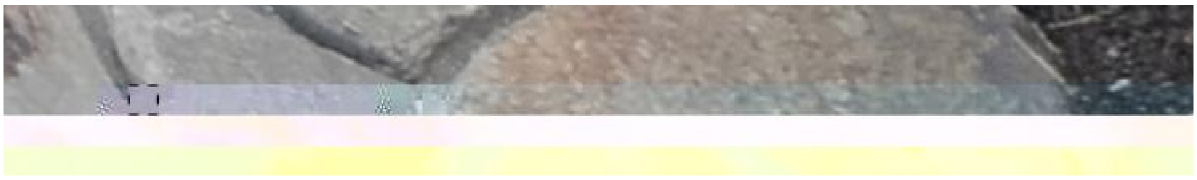


several extra MCUs are present so the rest of the image is not aligned; it is necessary to select bad MCUs (it’s easy to see that they don’t hold real image info) and delete them. Press ***s*** and use left/right keys to shift the region past the selected MCU until features come into alignment; notice that the status text informs you of how much the region is shifted:



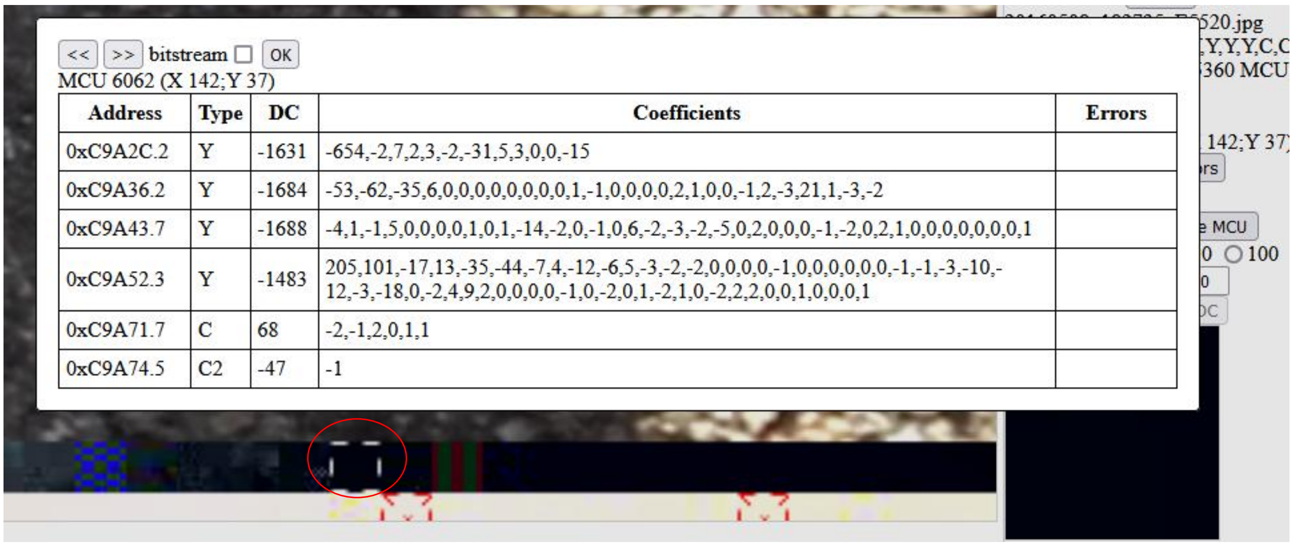
```
Selected MCU11288 (110;69)
Shift rendering by -8 MCU
xy(1469,1272), MCU(91;79)
Zoom:4.90
```

Press ***s*** again and select the same amount of bad MCUs; delete them. Now features are aligned:



Don’t worry about colors: they will be fixed later.

Proceed with the next corruption point; select one of the dark MCUs and click *Extra MCU info*:



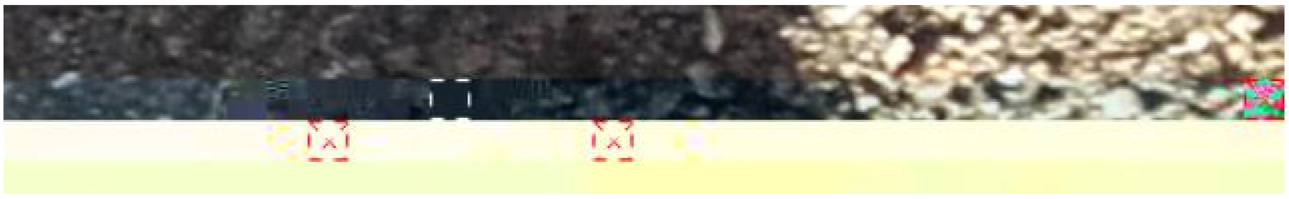
MCU 6062 (X 142;Y 37)

Address	Type	DC	Coefficients	Errors
0xC9A2C.2	Y	-1631	-654,-2,7,2,3,-2,-31,5,3,0,0,-15	
0xC9A36.2	Y	-1684	-53,-62,-35,6,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,2,1,0,0,-1,2,-3,21,1,-3,-2	
0xC9A43.7	Y	-1688	-4,1,-1,5,0,0,0,0,1,0,1,-14,-2,0,-1,0,6,-2,-3,-2,-5,0,2,0,0,0,-1,-2,0,2,1,0,0,0,0,0,0,0,1	
0xC9A52.3	Y	-1483	205,101,-17,13,-35,-44,-7,4,-12,-6,5,-3,-2,-2,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-3,-10,-12,-3,-18,0,-2,4,9,2,0,0,0,0,-1,0,-2,0,1,-2,1,0,-2,2,2,0,0,1,0,0,0,1	
0xC9A71.7	C	68	-2,-1,2,0,1,1	
0xC9A74.5	C2	-47	-1	

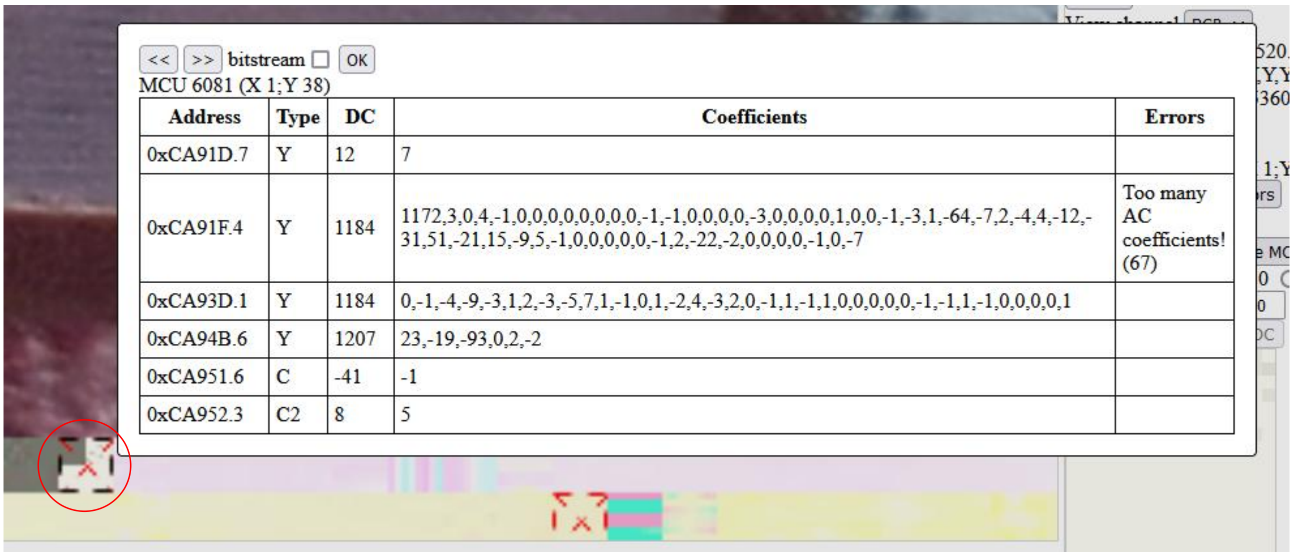
That DC level at -1631 looks really low, that’s why the MCU appears black; it is the result of -654 as DC coefficient (remember that a DC coefficient is relative to the previous DC level).

This is almost certainly a bad MCU, it can be deleted along with a few others around it.

After removing MCUs that have extreme DC coefficients, colors and detail reappear on subsequent MCUs. Follow the alignment procedure until the whole line appears to be in the original position:



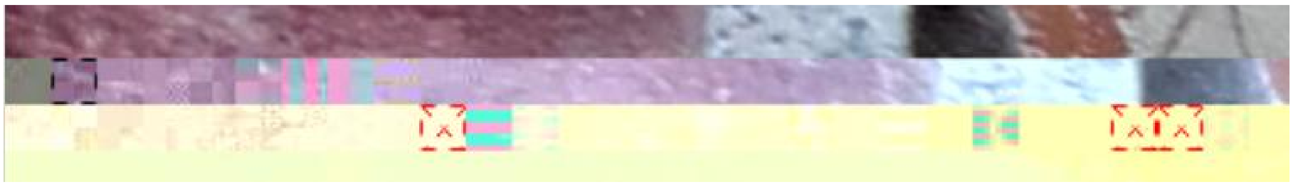
Next we can focus on getting some details back, as the colors are too saturated to allow aligning effectively. MCU 6081 is a good candidate: it has a decoding error (is red crossed) and colors change dramatically within it; let's examine it in detail:



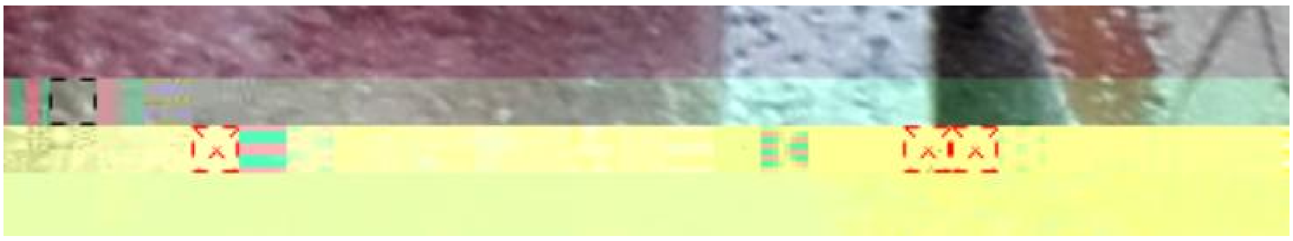
MCU 6081 (X 1;Y 38)

Address	Type	DC	Coefficients	Errors
0xCA91D.7	Y	12	7	
0xCA91F.4	Y	1184	1172,3,0,4,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,-3,0,0,0,0,1,0,0,-1,-3,1,-64,-7,2,-4,4,-12,-31,51,-21,15,-9,5,-1,0,0,0,0,0,-1,2,-22,-2,0,0,0,0,-1,0,-7	Too many AC coefficients! (67)
0xCA93D.1	Y	1184	0,-1,-4,-9,-3,1,2,-3,-5,7,1,-1,0,1,-2,4,-3,2,0,-1,1,-1,1,0,0,0,0,0,-1,-1,1,-1,0,0,0,0,1	
0xCA94B.6	Y	1207	23,-19,-93,0,2,-2	
0xCA951.6	C	-41	-1	
0xCA952.3	C2	8	5	

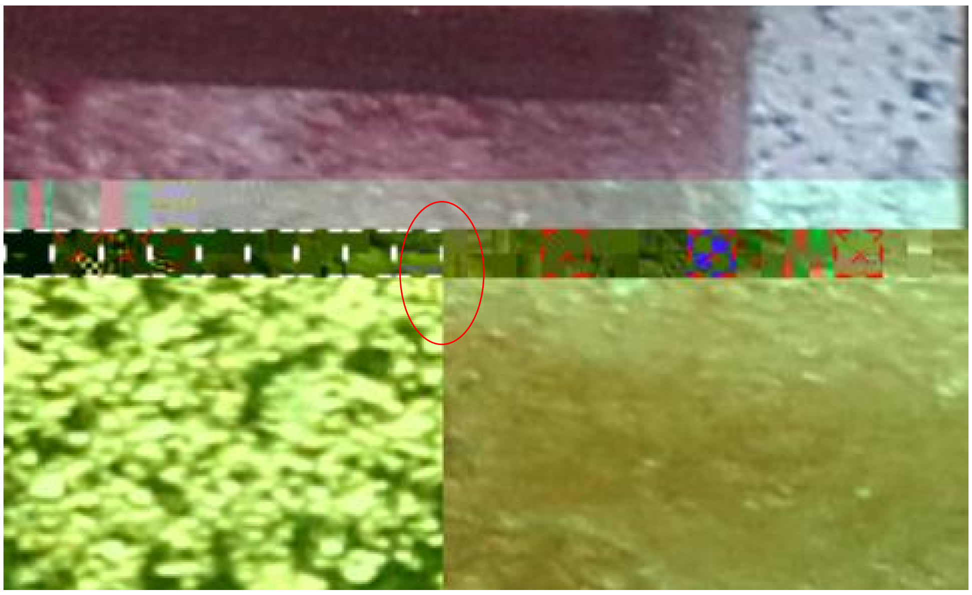
As expected, one of the Y blocks has a suspiciously high DC coefficient that causes a high brightness value; it has to be deleted.



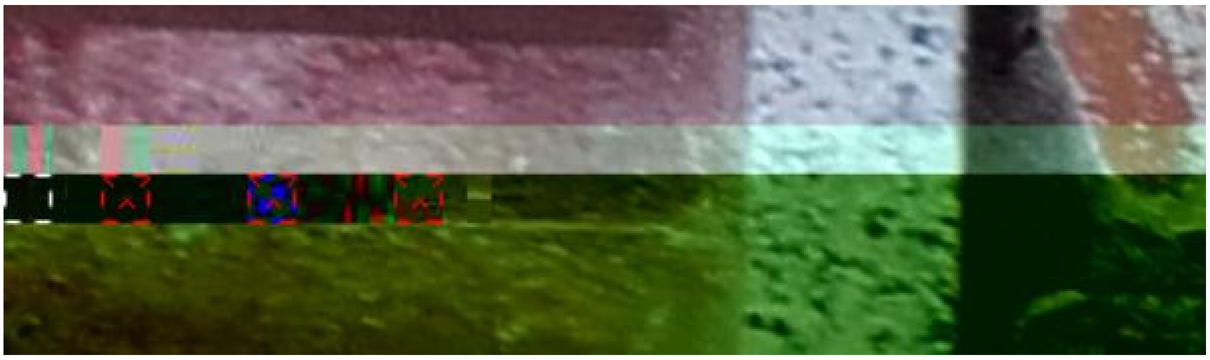
Much better now! Details are back and we can once again align MCUs against the previous line:



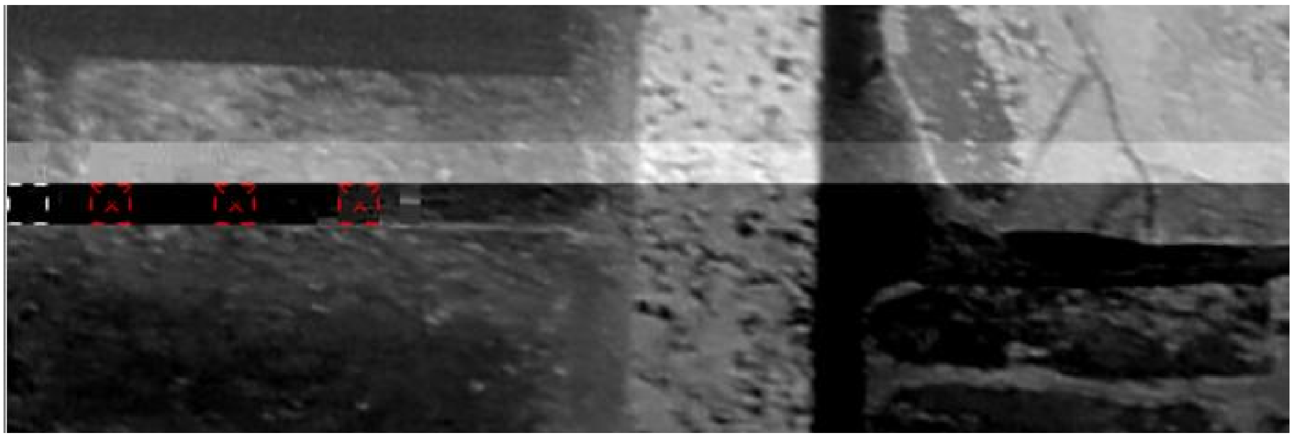
Sometimes it's easy to spot missing alignments, especially at an image border:



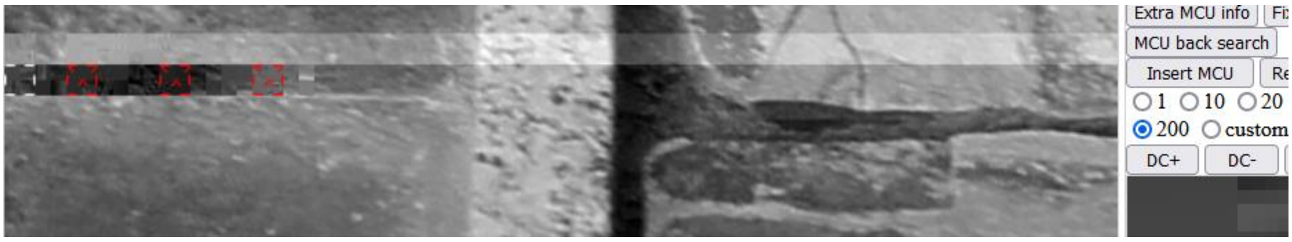
Deleting those 9 MCUs shifts the rest into perfect alignment:



Other than deleting MCUs, another way of correcting extreme brightness is by changing the DC level manually. You have first to select a single channel view, typically Y:

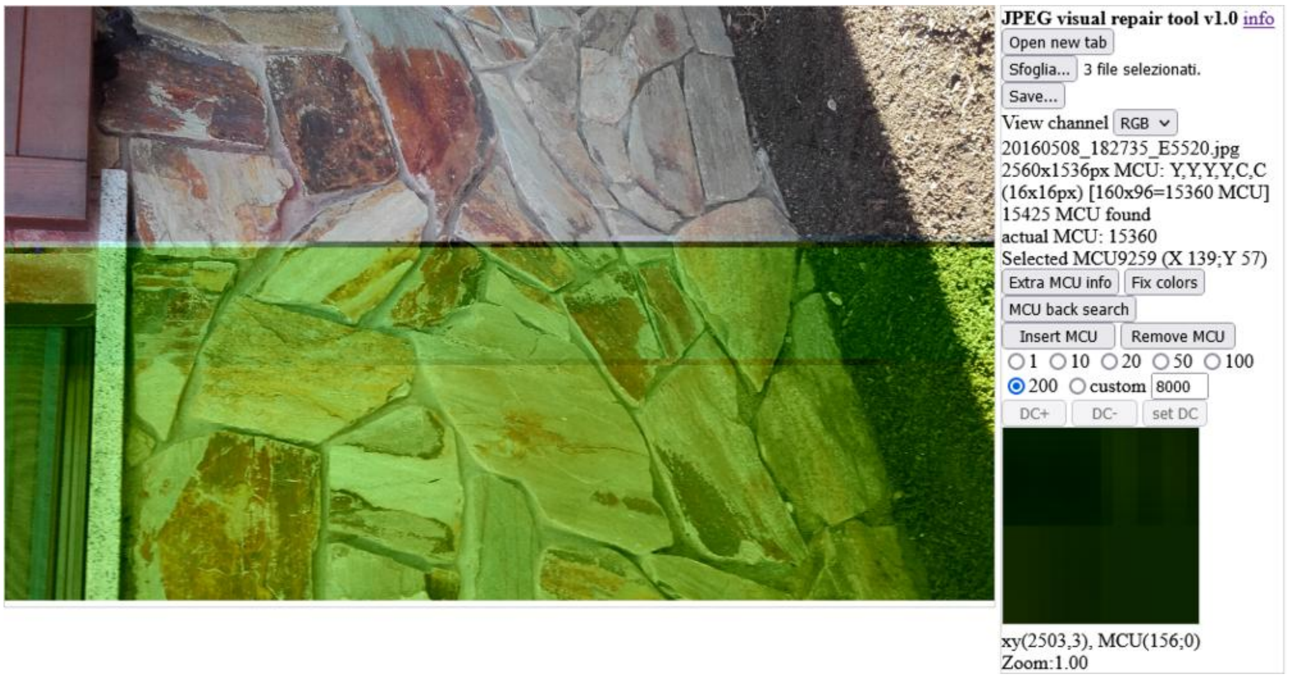


Now the DC+ and DC- buttons are enabled; we can try increasing by 200 until satisfied with the brightness:



1 10 20 [✓ 200] custom

A combination of the techniques described above can be used to roughly correct brightness and align the entire image:



3 file selezionati.

View channel RGB

20160508_182735_E5520.jpg
2560x1536px MCU: Y,Y,Y,Y,C,C
(16x16px) [160x96=15360 MCU]
15425 MCU found
actual MCU: 15360
Selected MCU9259 (X 139;Y 57)

1 10 20 50 100 [✓ 200] custom 8000

xy(2503,3), MCU(156;0)
Zoom:1.00

Color correction

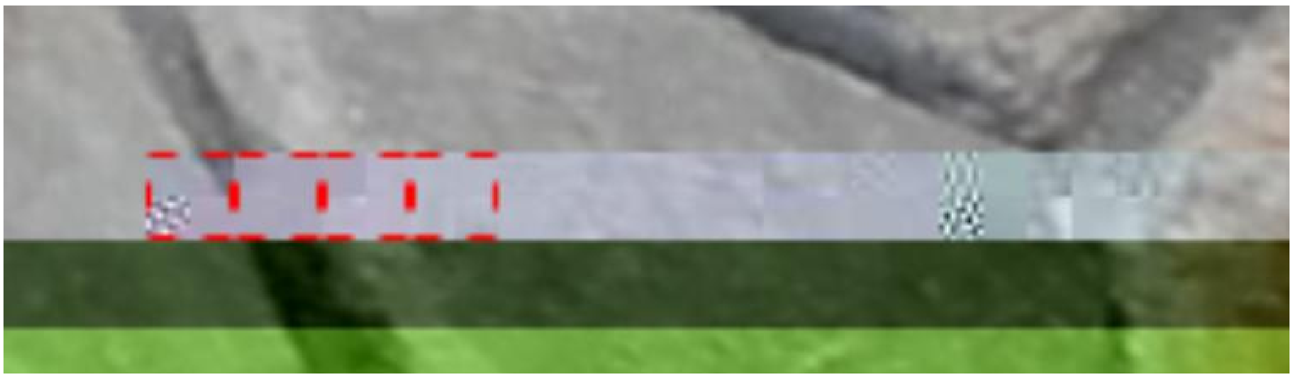
Although it would be possible to manually change DC level in order to reduce color differences, using the *Fix colors* function achieves the same results much faster.

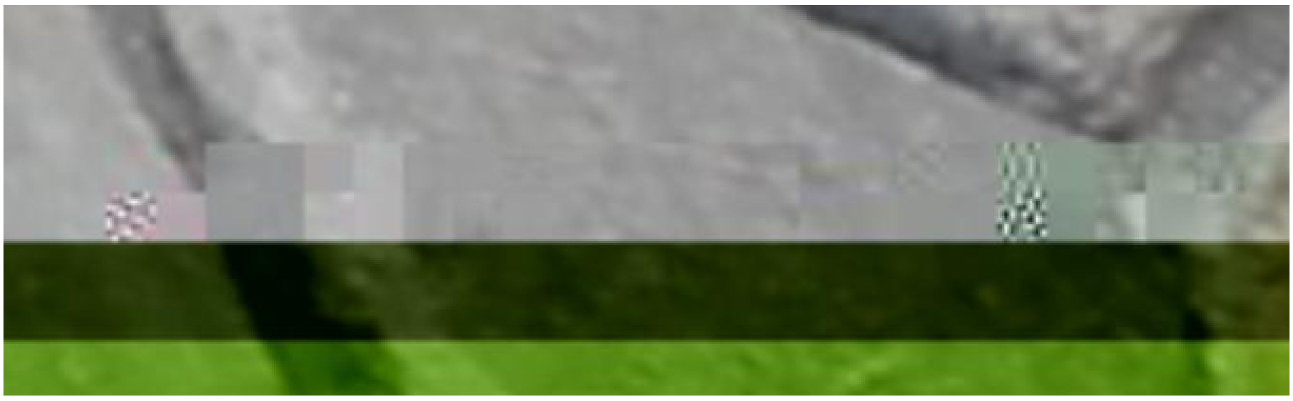
Color fixing is performed MCU by MCU, minimizing color difference between the top pixels row and the bottom row of the corresponding pixels in the previous line.

Select MCU by right clicking or dragging with the right button, then click on *Fix colors.*

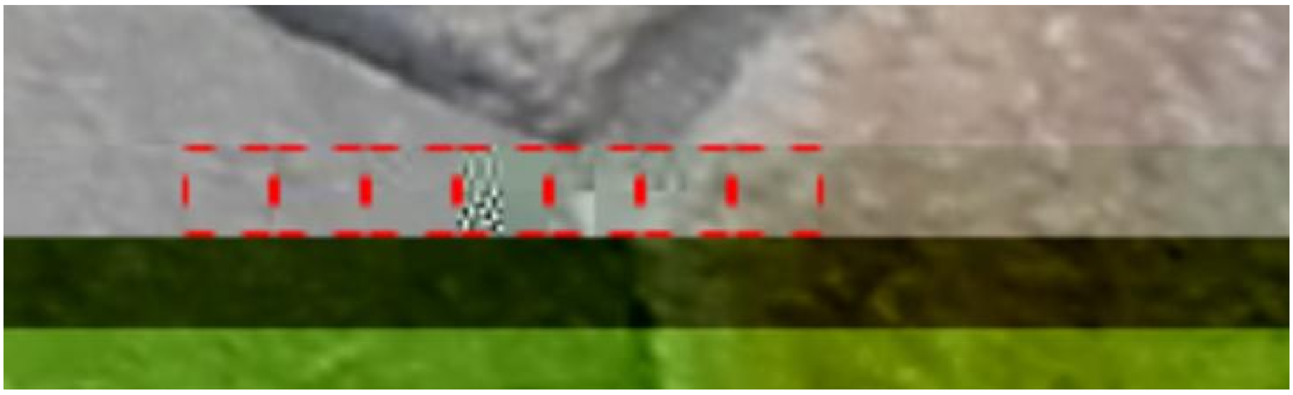
MCUs that are clearly damaged should be selected because their DC coefficients are likely bad; however, an area with undamaged MCUs (i.e. with plausible details that continue across boundaries) only requires correction on the first MCU: the rest have the right amount of DC shift.

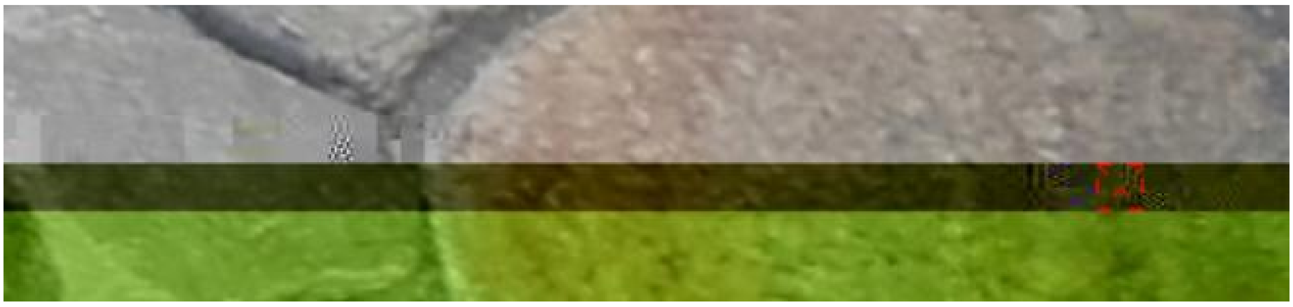
Below an example of *Fix colors, before and after*:



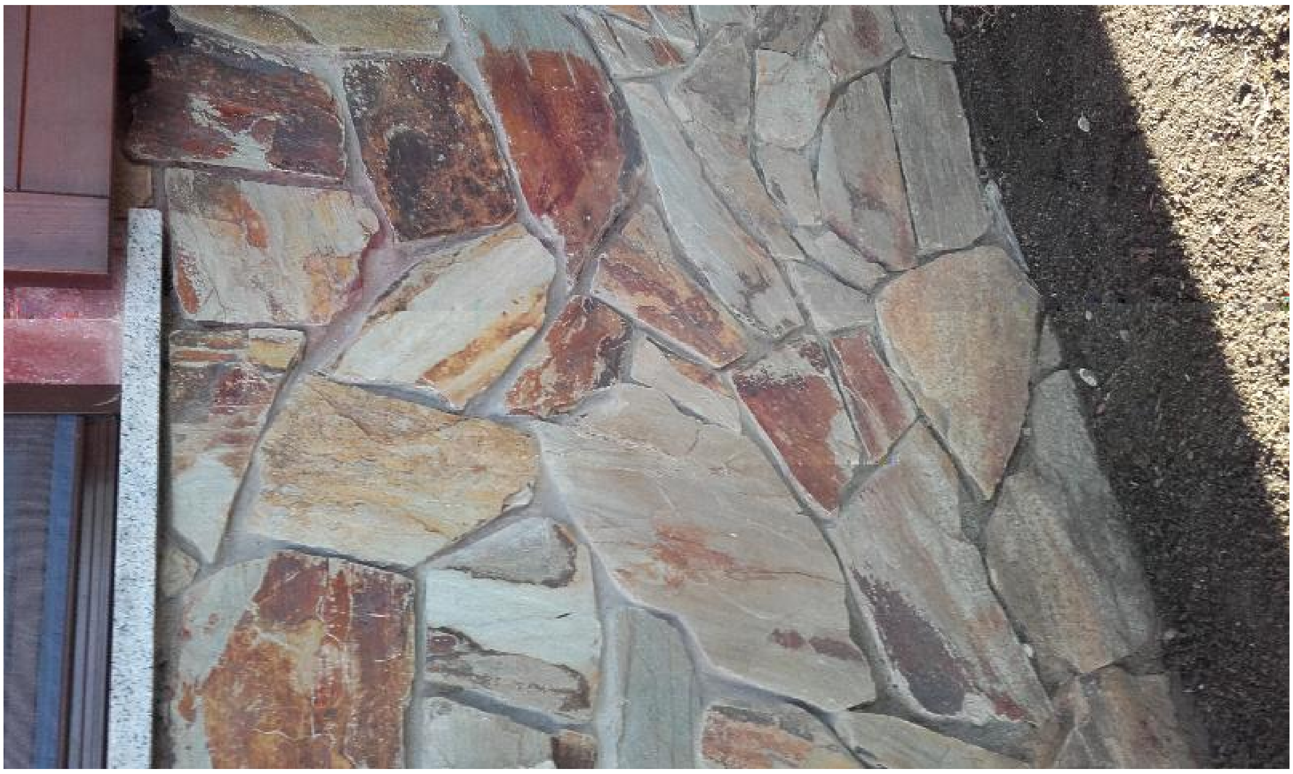


Yet another example:





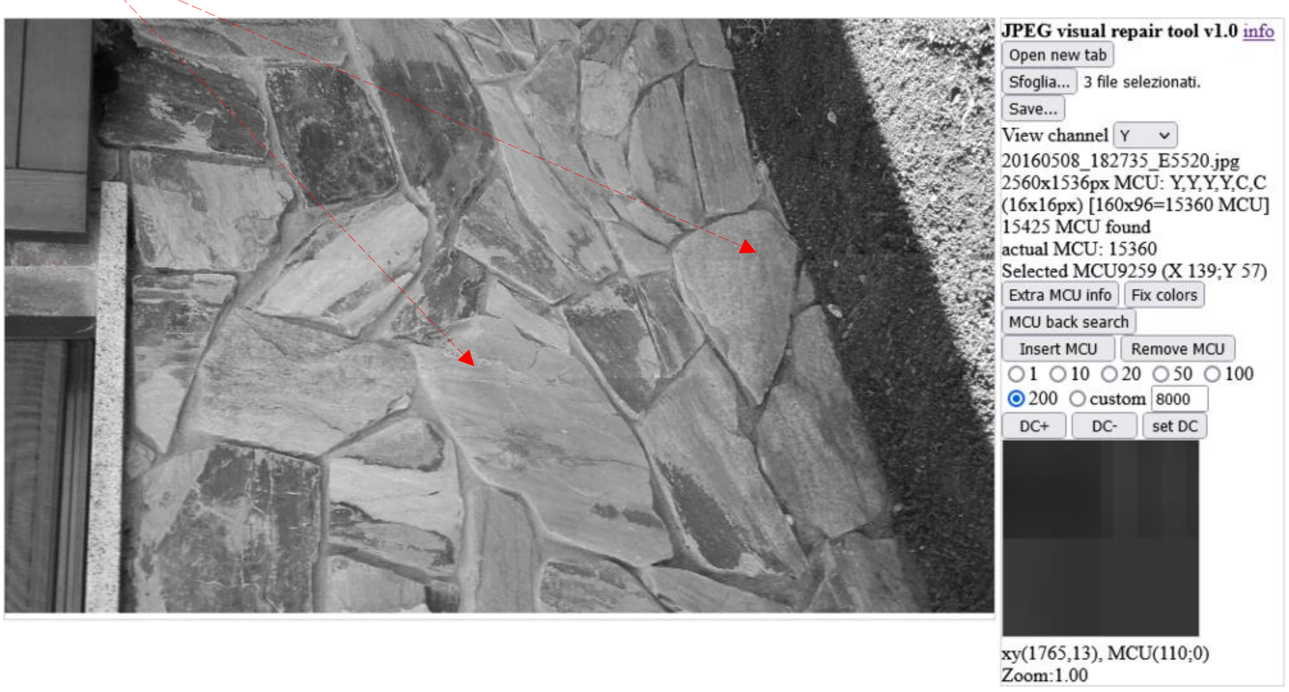
After applying the same procedure to all affected areas, this is the result:



Refinements

Although *Fix colors* usually does a good job of minimizing color differences, there are some cases in which color bands are still visible; when this happens it is necessary to change DC level manually.

Switch to a component view, like channel Y, and examine the image: find the exact origin of banding.


JPEG visual repair tool v1.0 info

Sfoglia... 3 file selezionati.

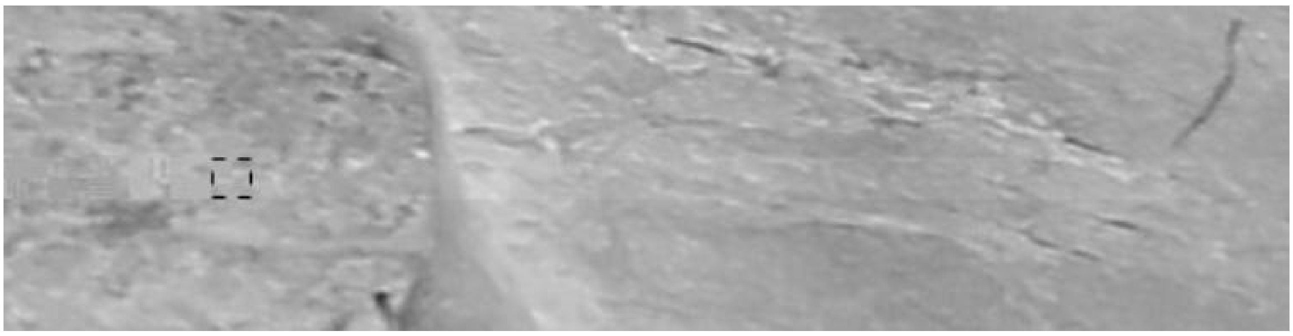
View channel Y

20160508_182735_E5520.jpg
2560x1536px MCU: Y,Y,Y,Y,C,C
(16x16px) [160x96=15360 MCU]
15425 MCU found
actual MCU: 15360
Selected MCU9259 (X 139;Y 57)

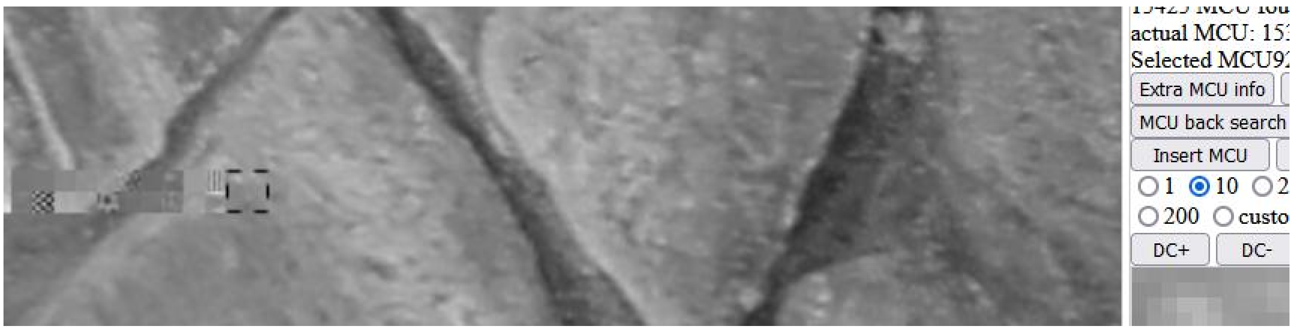
○ 1 ○ 10 ○ 20 ○ 50 ○ 100
[✓ 200] ○ custom 8000

xy(1765,13), MCU(110;0)
Zoom:1.00

Here an enlarged view:

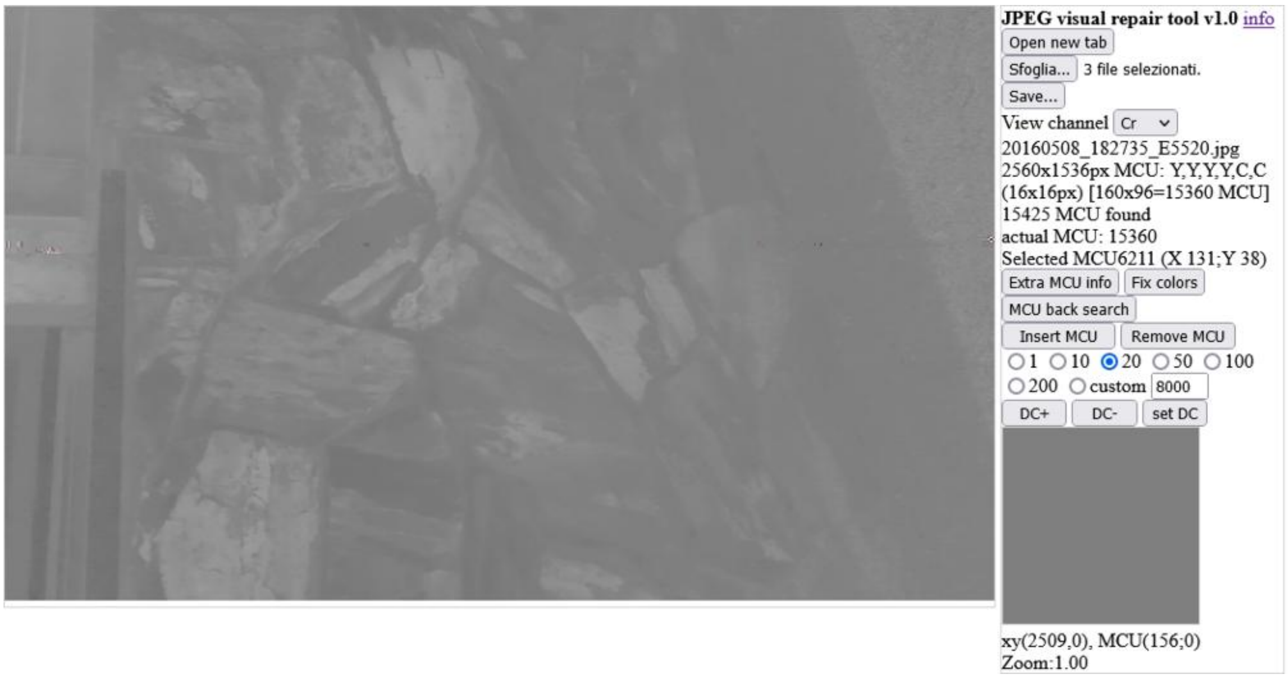


Playing with DC+, DC-, and the selected amount, it is possible to reduce the banding effect:


actual MCU: 15
Selected MCU9

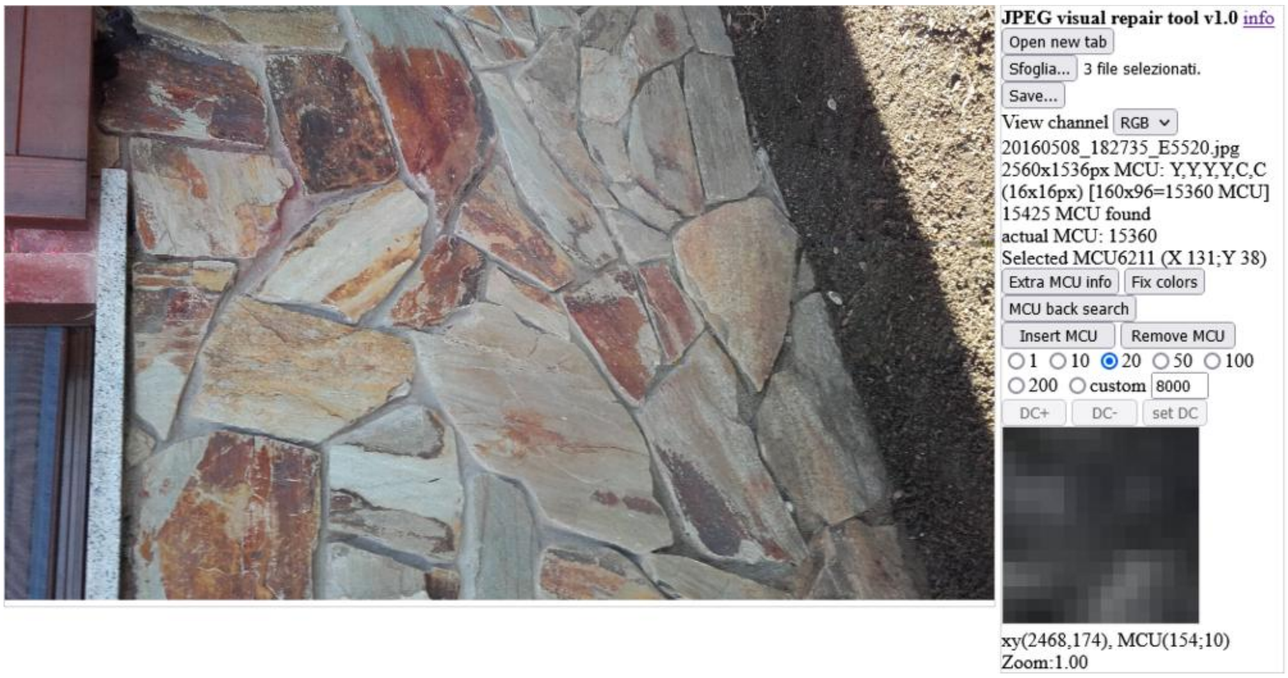
○ 1 [✓ 10] ○ 2
○ 200 ○ custo

The same thing should be done for the other channels, Cb and Cr:



3 file selezionati.
View channel Cr
20160508_182735_E5520.jpg
2560x1536px MCU: Y,Y,Y,Y,C,C
(16x16px) [160x96=15360 MCU]
15425 MCU found
actual MCU: 15360
Selected MCU6211 (X 131;Y 38)
○ 1 ○ 10 ◉ 20 ○ 50 ○ 100
○ 200 ○ custom 8000
xy(2509,0), MCU(156;0)
Zoom:1.00

In the end the image will appear correct on most of the area:



3 file selezionati.
View channel RGB
20160508_182735_E5520.jpg
2560x1536px MCU: Y,Y,Y,Y,C,C
(16x16px) [160x96=15360 MCU]
15425 MCU found
actual MCU: 15360
Selected MCU6211 (X 131;Y 38)
○ 1 ○ 10 ◉ 20 ○ 50 ○ 100
○ 200 ○ custom 8000
xy(2468,174), MCU(154;10)
Zoom:1.00

After saving, a final touch with a standard image editing program will correct the remaining artifacts due to damaged MCUs.

Further advice

It is often possible to read the same file from damaged media using different tools; multiple versions may differ and be damaged in different areas. Open all versions in different tabs (just select all of them in the *open file* dialog); combine good regions by using copy and paste MCU.

Anyways, never delete the original damaged file; a new tool may come up in the future that is able to correct even more errors.

(Essential) Introduction to the JPEG format

JPEG images are generated from bitmap images by removing some amount of detail according to a quality setting. This operation is performed on elementary blocks of 8x8 pixels.

Essentially, pixel data is transformed into frequency coefficients in a process known as 2D DCT; these coefficients are reduced in size in a way that preserves the perceived quality, then stored in a compressed form.

How exactly this is done is not essential for our purpose, however we have to understand a few additional details:

- pixels are first transformed from RGB to YCbCr; this means that luminance (Y, i.e. the gray scale representation) is coded separately from chrominance (Cb&Cr, i.e. color information).
- Y and C can be (and often are) sampled at a different resolution; that's because the human eye is more sensitive to luminance than chrominance. So the image is further subdivided in Minimum Coded Units comprising a certain amount of Y and C blocks, usually with more Y than C.

  Example: YYYYCbCr; this has 4 times as many Y as Cx; so the MCU is 2x2 blocks or 16x16 pixels, with full Y data and Cb/Cr sub-sampled at ½ the resolution in width and height. Other schemes are also possible.
- Y and C blocks are stored as a list of coefficients, with the first one, the DC coefficient, depending from the DC of the previous block; in fact it is coded as the increment from the previous one. Note that DC levels are chained per-component: all Y depend from each-other, all Cb together, all Cr together as well.
- Coefficients are compressed using Huffman coding (also rarely Arithmetic coding), resulting in strings of bits of any size. This means that it's not easy to understand where one coefficient ends, and also that editing cannot be done byte by byte. That is the reason why in general a hex editor is not useful for modifying or correcting an image.
- The layout of a JPEG image is divided into sections, separated by markers that are byte-aligned and begin with 0xFF. Each describes an aspect of the image, like size/organization, EXIF data, quantization tables, compression tables, etc. Stream data (the variable bit coefficients that represent the image) could also contain bytes at 0xFF, in which case a 0x00 byte is appended (this is called bit stuffing).
- In order to increase robustness against data loss, some images make use of Restart Markers. These markers are inserted every N MCU (a configurable number), and signal a break in the DC level chain; the subsequent block will code its DC level as a pure number and not as a delta with respect to the preceding block. In this way, any DC error will propagate at most up to the next restart marker.

There are may more details that are outside the scope of this guide and not essential for our purpose; they are nonetheless very interesting; here some links:

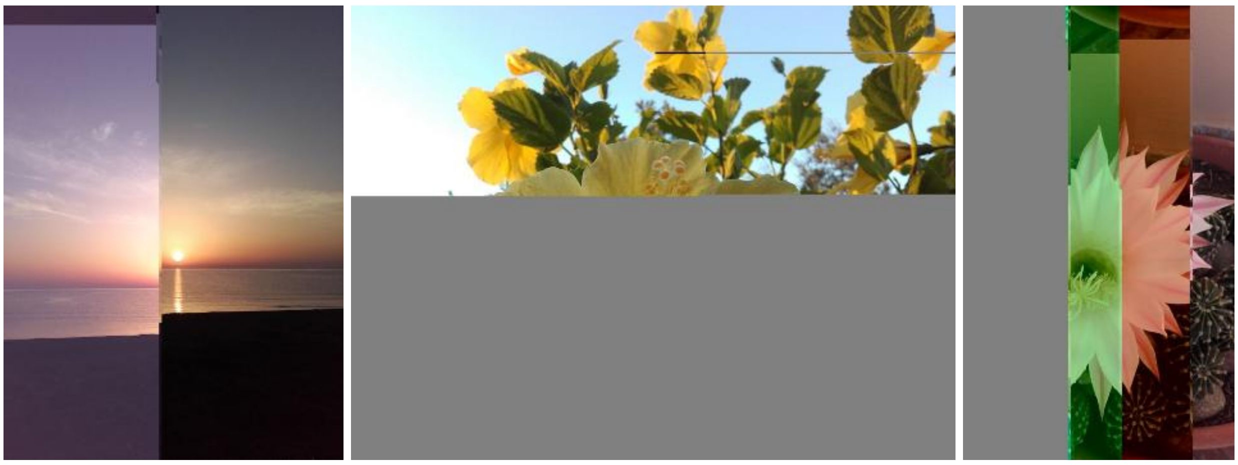
https://en.wikipedia.org/wiki/JPEG

www.impulseadventure.com/jpeg-compression (from WebArchive, as it recently went offline)

JPEG image corruption

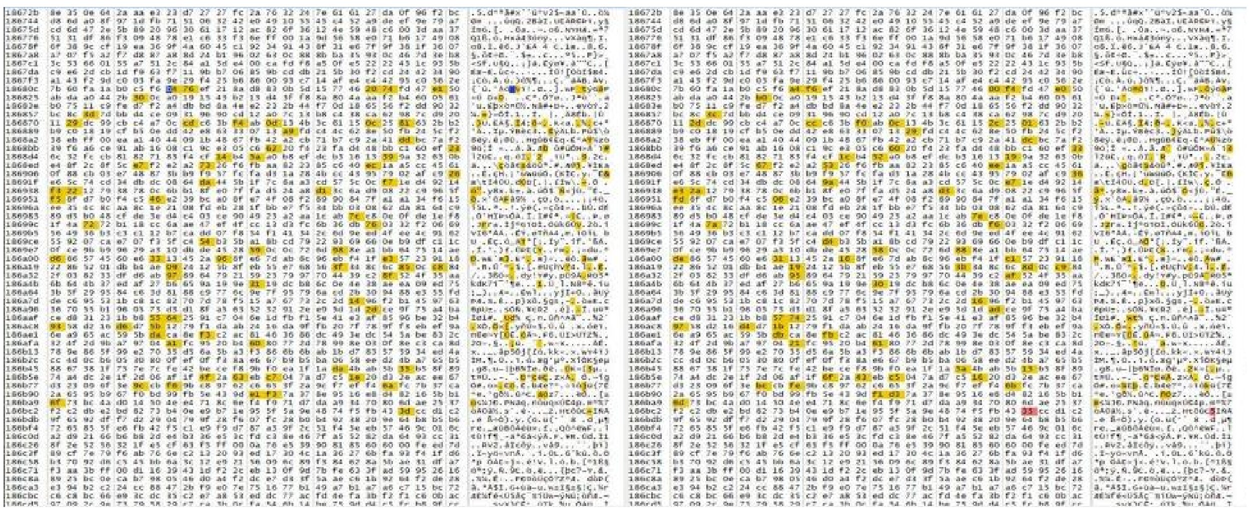
Image corruption has always been present, but lately it is becoming more common for a number of reasons: a lot more pictures are generated today; pictures are getting larger; storage media is often a memory card, which is inherently less reliable than an HDD or SSD.

Examples of JPEG image corruption: false colors, misalignment, missing sections.



Notice that corruption starts at a precise point in the XY scan, and proceeds from there until the end; frequently more than one corruption point is present.

Bit errors are often localized in a small area of the image file; not all bits and bytes are affected; as an example, below a comparison between a corrupted file and the original version.



There is a certain probability that changing even a single bit in the stream, a compression code prefix (Huffman code) is changed; the resulting code may signal a different bit length than the original one, therefore not only that coefficient is affected, but also the subsequent one, and so on.

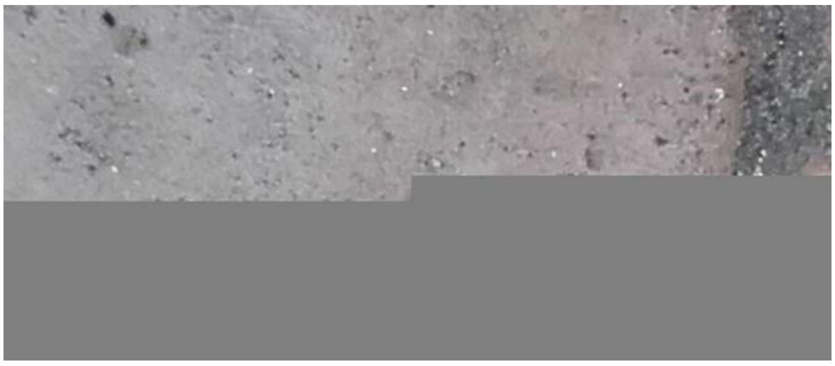
Fortunately it is very common that at some point after the error, and it may be many blocks down the line, the decoding process recovers the correct coefficient ordering; I don’t know the reason, it’s probably some kind of mathematical property of the Huffman coding.

Anyways, between the error occurrence and the recovery point we are left with a certain number of blocks that show more or less random coefficient values; the interpretation of this data by the decoding algorithm may lead to various artifacts.

DC level errors

These are due to the fact that a DC coefficient is expressed as variation with respect to the previous one; therefore, once a DC coefficient is corrupted, all subsequent blocks, even when free of errors, will result into wrong values.

For example, let’s suppose that as a consequence of a bit error the DC coefficient for a Y block is interpreted as 1500 instead of 10; this means that the luminance value will be extremely high and possibly will saturate the output range; regardless of the other (AC) coefficients in the same 8x8 block, the block will be rendered with a uniform color (exactly what color is codec-dependent).



The subsequent block will start from the same brightness level (plus its own DC value); it will likely be rendered as out of range as well. The same happens from now to all remaining blocks, hence the image appears of a uniform color from this point on.

It is interesting to note that, save for the damaged block, all pixel information is still present but is simply rendered incorrectly.

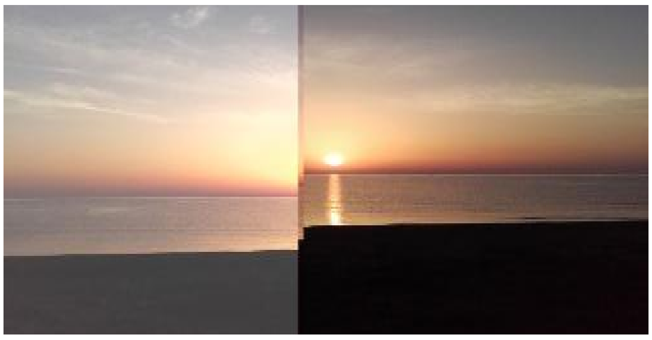
The same type of corruption can result into various degrees of high or low brightness or, if the error happens to Cb/Cr blocks, weird colors can appear.

It is actually very common that more than one block is affected at the same time, so both Y and Cb/Cr get corrupted.

Alignment errors

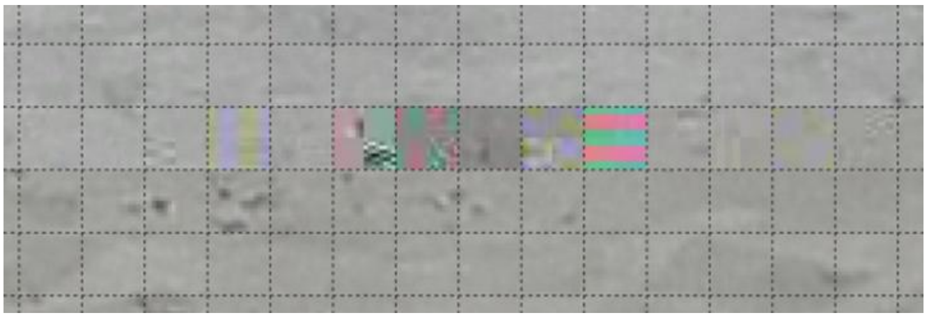
A possible and frequent effect of bit errors is that blocks are terminated early; this is somewhat codec-dependent, as the JPEG standard does not describe the behavior in case of impossible situations like an undeclared Huffman code or more than 64 coefficients. Anyways, it is often the case that in place of a number of original blocks, the decoder produces more of them, also disrupting the sequence between Y and C (note that there is no way to distinguish them other than counting).

By the time the decoder recovers the correct sequence and block order (if it does!), the rows will certainly be misaligned, like in the image below:



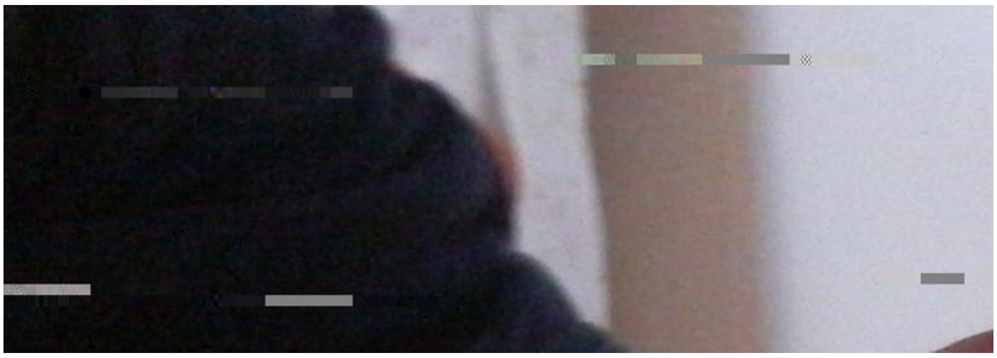
AC errors

AC coefficients in a block define the shapes present, i.e. the fine detail. A corrupted AC coefficient may be visible or not, depending on its value. A small level of error checking is given by the hard limit of 64 coefficients per block. However, the highest impact of an AC coefficient error is that it can affect all subsequent coefficients if its length is decoded incorrectly; this can even extend to subsequent blocks, and theoretically to a whole image (although it is very improbable).



Errors with Restart markers

When present, restart markers limit the extent of artifacts to a finite number of MCU; however it is always possible that the restart marker itself is corrupted. In these cases, most codecs ignore data until the next correct marker is found, so the only visual effect is that parts of lines are of a uniform color; the overall shape is still correct and also colors are unchanged.



Header corruption

The first part of a JPEG image contains various sections that specify how to interpret stream data.

Errors in this area will generally result in unreadable images.

Copying the header from a similar but readable image should fix the issue.