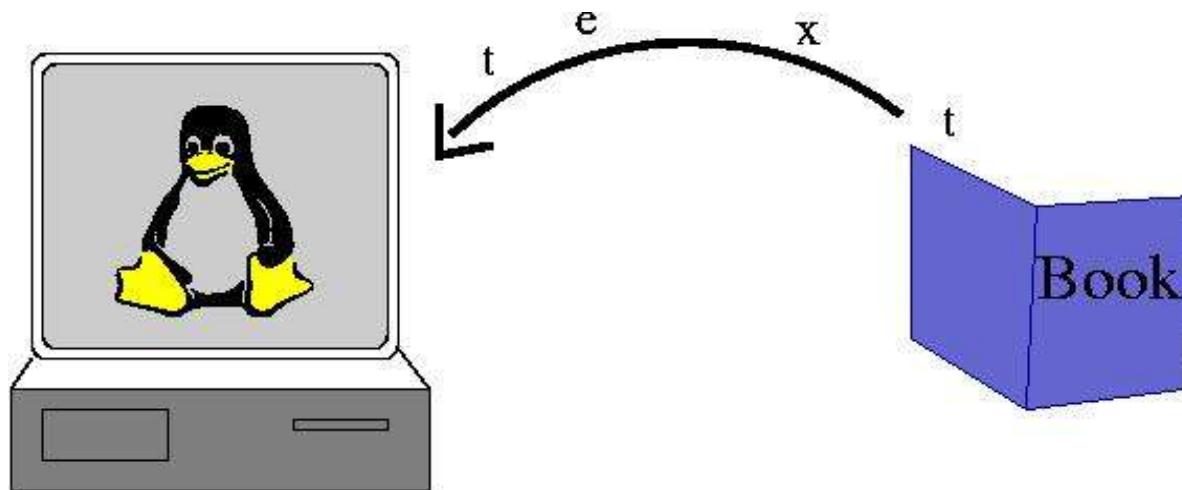


GOOCR

Optical Character Recognition

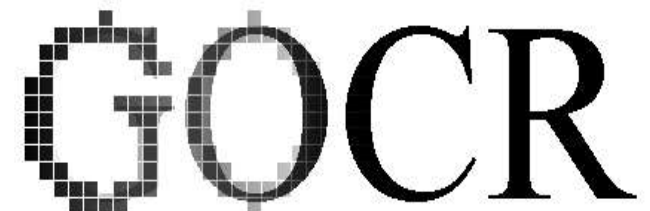


GOCR, how to use it?

- **Ask your questions everytime!**
- how does it work (short)
- examples + tips and tricks
- tuning and coding
- questions + diskussion

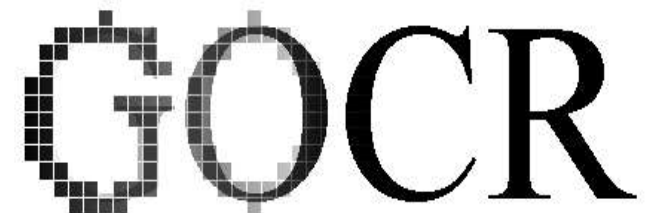
How does it work?

- preprocessing
 - threshold value detection
 - box detection, zoning, line detection
 - sorting and melting, dust, pictures, ...
- call ocr engine (3 engines, 2 experimental)
 - repeated for unknown chars
- postprocessing (XML, TeX, UTF, ASCII)

The logo for GOOCR, where the 'G' is rendered in a pixelated, blocky font, and the 'OOCR' is in a standard serif font.

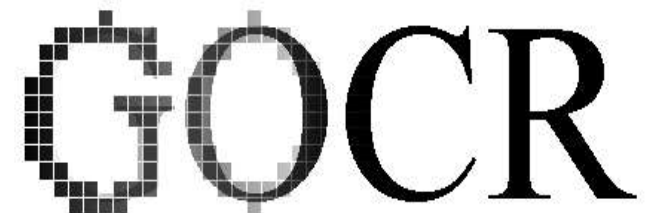
running GOCR

- `gocr -h` # short man page
- `gocr sample.jpg` # best case scenario
- `gocr -m 130 sample.jpg` # database
- `gocr -v 39 -m 58 -e - sample.jpg`
debugging + out30.png

The logo for GOCR, where the 'G' is rendered in a pixelated, blocky font, and the 'OOCR' is in a standard serif font.

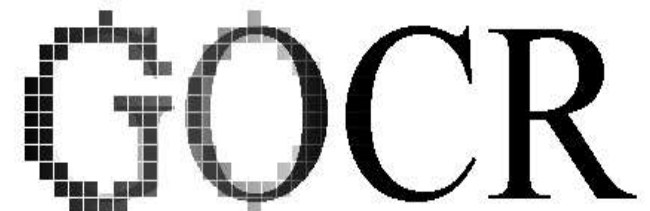
Looking inside ...

- assume no **colors**, black on white only
- assume no rotation, same *font*, all characters are `s e p a r a t e d`
- try to repair if assumptions are hurt (can fail)
- every char is recognized empirically based on its pixel pattern
- lot of possibilities to improve GOCR

The logo for GOCR, where the 'G' is rendered in a pixelated, grid-like font, and the 'OOCR' is in a standard serif font.

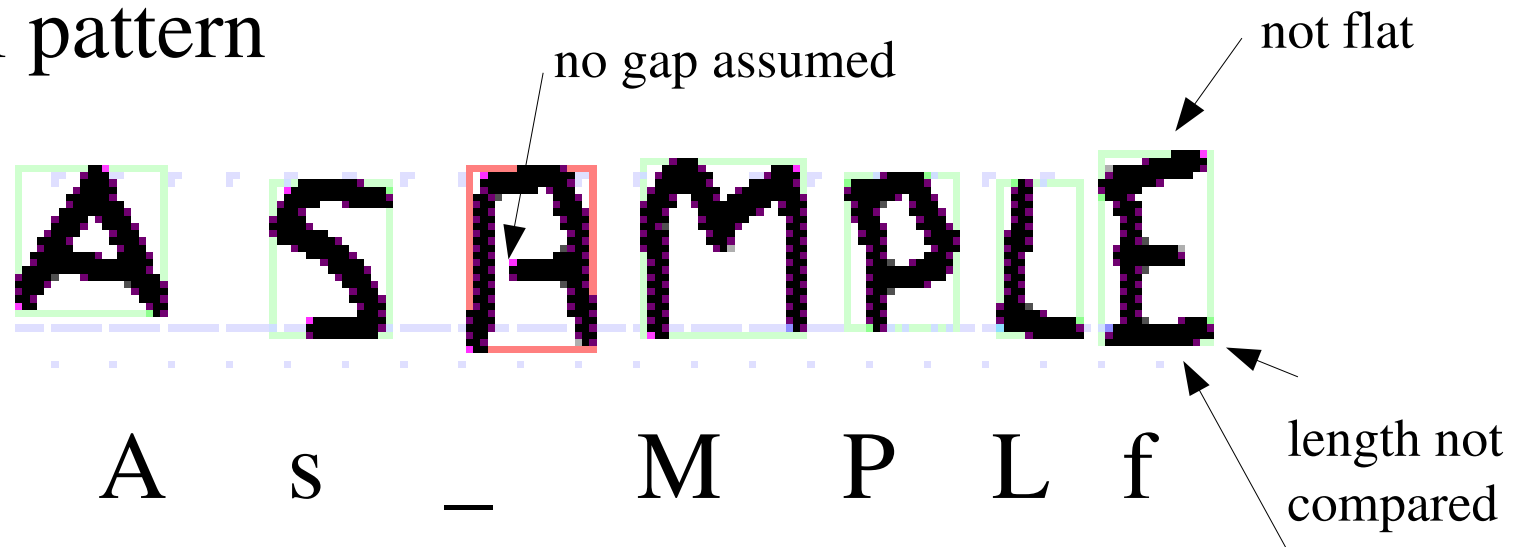
Looking inside ...

- pgm2asc (pgm2asc.c)
 - otsu + thresholding (otsu.c), load_db (database.c), scan_boxes
 - remove_dust (remove.c), remove_borders, detect_barcode
 - detect_pictures, detect_rotation_angle, detect_text_lines, ...
 - [char_recognition](#) (adjust_text_lines, char_recognition)
 - compare_unknown_with_known_chars
 - try_to_divide_boxes, list_insert_spaces, context_correction

The logo for GOOCR, where the 'G' is rendered in a pixelated, grid-like font, and the 'OOCR' is in a standard serif font.

A sample

- every char is recognized empirically based on its pixel pattern



spacing failed, (upper)case failed, unknown char, wrong char

verbose GOCR (-v 39)

status:

```
# Optical Character Recognition --- gocr 0.40  
# options are: -l 0 -s 0 -v 39 -c f -m 0 -d -1 -n 0  
vortrag/sample.png
```

```
# using unicode
```

reading file:

```
# popen( pngtopnm vortrag/sample.png )  
# readpam: format=0x5036=P6 h*w(d*b)=45*189(3*1)  
# readpam: min=0 max=255
```

reading database:

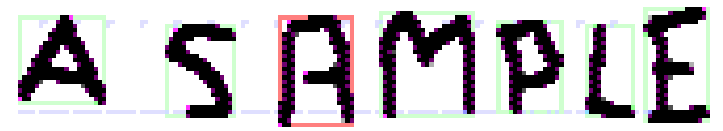
```
# db_path= (null)
```

threshold value detection:

```
# OTSU: thresholdValue = 81 gmin=0 gmax=255  
# thresholding new_threshold= 160
```

box detection:

```
# scanning boxes nC= 9 avD= 15 20
```



A SAMPLE

verbose GOCR (-v 39)

dust detection and smoothing:

```
# auto dust size = 0 nC= 9 avD= 15 20  
# ... 0 white pixels removed, cs=160 nC= 9  
# smooth big chars 7x16 cs=160 ... 0 changes in 7 of 9
```

special objects:

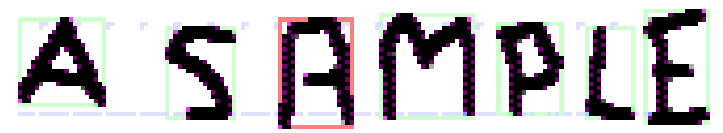
```
# detect barcode , 0 bars, boxes-0=9  
# detect pictures, frames, noAlphas, mXmY= 15 20 ... 0 - boxes 9  
# averages: mXmY= 15 20 nC= 9 n= 9
```

clean the border:

```
# remove boxes on border pictures= 0 rest= 9 numC= 9  
# ... deleted= 0, within pictures pictures= 0 rest= 9 numC= 9  
# .... deleted= 0, pictures= 0 rest= 9 numC= 9
```

some alpha code (new rotation angle detection):

```
# rotation angle (x,y,num) (23990,731,7) (0,0,0), pass 1  
# rotation angle (x,y,num) (23990,731,7) (68352,256,4), pass 2
```



A S A M P L E

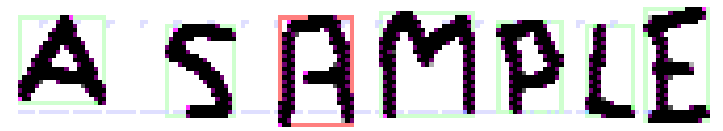
verbose GOCR (-v 39)

line detection:

```
# detect longest line - at y=0 crosses= 0 my=0 - at crosses= 0 dy=0
# scanning lines
# ... trouble on line 1, wt/%= 56
# bounds: m1= 7 m2= 2 m3= 22 m4= 24 my= 20
# counts: i1= 4 i2= 3 i3= 2 i4= 5
# all boxes of same high! num_lines= 1
# add line infos to boxes ... done
# mark lines
```

box corrections:

```
# divide vertical glued boxes, numC 9
# searching melted serifs ... 0 cluster corrected, 0 new boxes
...
# glue broken chars nC= 9
# ... 0 fragments checked, 0 fragments glued
# ... 2 holes glued (same=0), 0 rest glued, nC= 7
# detect dust2, ... 0 + 0 boxes deleted, numC= 7
```



A SAMPLE

verbose GOCR (-v 39)

pitch detection:

```
# check for word pitch
# ...WARNING measure_pitch: only 6 spaces
# overall space width is 2 proportional
```

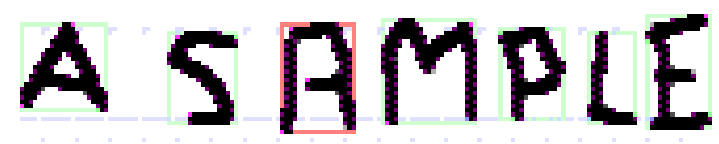
char recognition:

```
# step 1: char recognition unknown= 7 picts= 0 boxes= 7
# 3 of 7 chars unidentified
# adjust lines diff= 1
# step 1: char recognition unknown= 1 picts= 0 boxes= 7
# 2 of 7 chars unidentified
# debug: unknown= 1 picts= 0 boxes= 7
# mark lines
pnmtopng: 13 colors found
# step 2: try to compare unknown with known chars - found 0
```

A sample of text "A SAMPLE" where each character is enclosed in a colored bounding box. The 'A' is in a green box, 'S' is in a green box, 'A' is in a red box, 'M' is in a green box, 'P' is in a green box, 'L' is in a green box, and 'E' is in a green box. The text is rendered in a pixelated font.

verbose GOCR (-v 39)

```
# step 3: try to divide unknown chars
# divide box:    73    8  18  26
# list pattern x=  73    8 d=  18  26 t= 1 2
...@@@@@@@@@@@@... -
..@@@@@@@@@.@@@@...
.@@@.....@@@@..
.@@@.....@@@@.
.@@@.....@@@@.
.@@@.....@@@@.
.@@@.....@@@@.
.@@@...@@@@@@@@@.
.@@@..@@@@@@@@@@@@.
.@@@.....@@@@.
.@@@.....@@@@@
@@@@.....@@@
@@@.....@@@
.@@..... -
x123= 5 10 8  m123= 258 142 141
x c12 = 5 ( (?). ( 98) ( 0)
x c12 =10 (? ) (?). ( 0) ( 0)
x c12 = 8 (? ) (?). ( 0) ( 0), numC 7
```



verbose GOCR (-v 39)

```
# list shape      6 x= 160      6 d= 16   27 h=0  o=1 dots=0 0066 f
# list box dots=0 boxes=1 ... c=f mod=(0x00) line=1 m=3 4 24 29
# list box      x= 160      6 d= 16   27 r= 10 0 nrun=0
# list box frames= 1 (sumvects=44)
# frame 0  44 vectors= 10 0  9 1   6 1   5 2   2 2   0 4   0 5 ...
# list box char:  f(100)
# list pattern x= 160      6 d= 16   27 t= 1 2
.....$@$@@@@$.      .....@@@@@@@@@.
.@@@@@@@@@$@@$.    .@@@@@@@@@@@@@@@...<
$@@@@@.....      @@@@@@.....
..$@@$.          ..@@@.....
...$@@.....      ...@@@.....
...@@$.          ...@@@.....
..$@@@@$@@$.    ..@@@@@@@@@@@.....
..$@@@@$@@$.    ..@@@@@@@@@@@@@.....
...@@$.          ...@@@.....
...@@@.....      ...@@@.....
...@@@.....      ...@@@.....
...$@@@.....$@@$.  ...@@@@.....@@@@@.
.$@@@@@@@@@@@@@@$.  .@@@@@@@@@@@@@@@@@@@
```



verbose GOCR (-v 39)

spacing:

```
# insert space between words (dy=27) ... found 6
```

context correction:

```
# step 4: context correction I11 00
```

```
# store boxtree to lines ...
```

```
get_least_line_indent: page_width 189, dy 0
```

```
Line 1, y 8, raw indent 11, adjusted indent 11
```

```
Minimum indent is 11
```

```
... 2 lines, boxes= 6, chars= 6
```

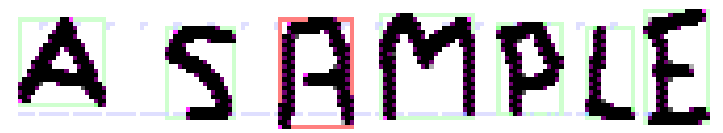
```
# debug: ( _ )= 1 picts= 0 chars= 6 (A)=1
```

```
# mark lines
```

```
pnmtopng: 12 colors found
```

```
A s _ M P L f
```

```
Elapsed time: 0:00:79.418.
```

A sample of text "A S A M P L E" where each character is enclosed in a small, colored rectangular box. The boxes are colored in a sequence: green, red, green, red, green, red, green. The text is rendered in a pixelated, monospace font.

Tuning

```
gocr -C A-Z sample.png # use optional filtering  
output: A _ _ M P L _ (E is unexpected)
```

```
gocr -m 130 -C A-Z sample.png # use database  
input: S A M (94%) L (74%) E  
output: A S A M P L E
```

```
gocr -s 13 -m 130 -C A-Z sample.png # pitch  
output: A SAMPLE
```

Debugging

- Normally its time to contact the author
 - sending him the sample and an explanation that 'E' is failing
- Also possible: Find out, why 'E' is not recognized.
 - setting C_ASK to 'E' in ocr0.c and recompile
 - rerun (-m 56)
 - now for every char ocr0.c explains, why its not a 'E'
 - output the break line in ocr0.c and the pattern
output: DBG L592 (160,6): break



A SAMPLE

Debugging

output: DBG L592 (160,6): break

```
591: for( x=dx,y=y0+dy/6; y<y1-dy/9; y++ ) // left border straight
592: { i=loop(box1->p,x0,y,dx,cs,0,RI); if( i>x+dx/9 ) Break;
593:   if(i<x) x=i;
594: } if( y<y1-dy/9 ) Break; // t
```

- Bug found! Break -> break; but L594 also printed out,
- code defines max. jump on left side of $dx/9$ ($dx=16$ here)
- its too restrictive for handwritten E's (we have max jump = 3)
- changing it to $dx/5$ could cause misdetections for other chars



A S A M P L E

Debugging

Why 'f' is recognized instead of 'E'?

- Searching for 'f' in ocr0.c
- add some code to distinguish between E and f
- leave the 'f'-subfuncion with a Break or reduce probability for the 'f'
- check, that 'f' of all other fonts are still recognized



A S A M P L E

The image shows the word "A S A M P L E" in a pixelated font. Each letter is enclosed in a thin, colored rectangular box. The boxes are colored as follows: 'A' (green), 'S' (green), 'A' (red), 'M' (green), 'P' (green), 'L' (green), and 'E' (green). This visualizes the character recognition process where each letter is identified and bounded.

Testing new code

- minimum test base is `./examples/*.png`
- results should never be worse than before patch code
- bigger public database is needed
- some automatition is desirable telling the programmer what has changed in the output