The Power of Ultracode

AIM Engage Again Conference

8 September 2021

Clive Hohberger & Terry Burton

Color Bar Codes – History Prior to Ultracode

- Early designers were infatuated with increasing data density by use of 16 to 64+ pixel colors in place of B/W
  - No symbologies in public domain prior to AIM Ultracode and ISO/IEC JAB Code
- Symbols were designed for printing only
  - Printing inks often fade, especially magenta: Affects red and blue hues
  - Light intensity and colour temperature affects perceived hues
  - Printed image scanning is very sensitive to lighting: Colour hues → grey in low light
- High resolution color digital cameras were new and expensive
- Ultracode *AIM Int'l Symbology Standard* (v1) issued in 2016
  - 10 year development, including the Members of entire AIM TSC

Ultracode Features

- Designed for CMYK color printing and sRGB electronic display
  - Optimized for smartphone symbol display and scanning
  - Scanning compensates for real-world changes in lighting of printed sysmbols
- Supports Unicode, 8-bit character sets and multibyte languages
  - Default encoding for ISO/IEC 8859-1 and Unicode UTF-8
  - Data compaction modes for all ISO/IEC 8859-n 8-bit characters and Unicode
  - Special URL compactions: Example: http://www encoded in 1 codeword
  - Chinese, Japanese and Korean byte-multibyte encoding
- Support for AIM ECI protocol and GS1 Digital Link Protocol
- Reed-Solomon Error Correction used in 2 new ways

Ultracode Color Bar Code Symbology (v2)

- Enabled by, and designed for the color smartphone
  - *Smartphone is now world's largest selling color barcode scanner (~7b)*
  - Mobile phones are increasingly used as data carriers instead of paper for one-time boarding & resort passes; event ticketing and access control
  - Computing power, display and camera resolution of today's smartphones enable 2D color barcodes
- 2021 consumers have expectations of *color everywhere* on phones, electronic displays and billboards
  - Black and white images mostly used for text display
  - The popular QR Code is a legacy B/W matrix symbol designed for printing

Comparing Ultracode and QR Code

- Ultracode is more spatially efficient, especially in encoding URLs
  - Ex: URL https://aimglobal.org/jcrv3tX encoded using same module size

- The same data encoded using the same module size and at a similar Error Correction level: The Ultracode symbol uses < ½ of the area of a QR Code symbol

Ultracode is a color bar code for personal devices

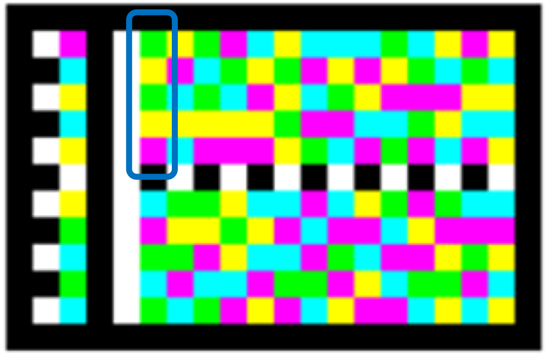
- Ultracode published by AIM in 2016; now being submitted to ISO
- Utilizes robust B/W internal finder patterns; colored data tiles
- Self-checking symbol characters almost double the Reed-Solomon error correction capability

2-row URL symbol

3-row Japanese baseball game ticket

Ultracode Technology

Ultracode symbol characters



- 2 to 5 rows of 5-module column codeword tiles
  - Each tile encodes a single value 0-282
- Encodes 283 values in each CMYG color tile
  - As compared with 32 values for 5 black/white pixels
  - Achieve ~9x increased data density for 5-pixel CMYG colour tiles
- Codeword tiles designed under rules to be *self-checking*
  - There must be at least 3 colors represented in each tile
  - There must be at least 1 Yellow or Green module in each tile
  - No codeword tile can have 3 modules the same color
  - No 2 vertically-adjacent modules may be the same color
- Tiles which fail the self-checking rules are treated as ***erasures*** in RSEC

Reed-Solomon EC use in Ultracode is special

- Reed-Solomon Error Correction is used in all 2D symbologies
  - Introduced in PDF-417
  - Theoretically, the most efficient form of mathematical error correction
- There are 2 kinds of corrections
  - **Errors**- Unknown data codeword error at an unknown location in symbol, require 2 RSEC characters to find and correct the erroneous codeword
  - **Erasures**- Unknown data codeword error at known location in symbol, require only 1 RSEC characters to correct the erroneous codeword
- Self-checking codewords detect damaged tiles as erasures at known locations rather than as errors at unknown locations
  - Can ideally double RSEC efficiency over other 2D symbologies

Ultracode uses a single GF(283) block

- GF(283) symbol characters allow encoding of 8-bit data stream and encoding commands in the same codeword stream
  - Codeword values 0-255 reserved for data
  - Codeword values 256-282 for encoding instructions, such as compactions
- With a 282-codeword block, Ultracode symbols typically have a lot of truncated codewords, due to efficient data compaction
- Truncation of unused data codewords done in Ultracode using a mathematical property of Reed-Solomon Error Correction process
  - Avoids the needs for padding out a big symbol with little data content
  - Truncation minimizes the Ultracode symbol size

AIM Extended Channel Interpretation (ECI)

- ECI in inherently included in Ultracode, but use is not mandatory.
  - Start Codeword 256 indicates a non-ECI message with default encoding ISO/IEC 8859-1.
  - Start Codeword 257 invokes ECI protocol starting with \000003 (ISO/IEC 8859-1 encoding), and data may subsequently include other ECIs.
  - Other Start Codewords invoke ECI protocol, e.g. CWs 258 - 271 => \000004 – \000018 (8-bit Latin-N) and CW 279 => \000026 (UTF-8 encoded Unicode).
  - Explicit indication of non-language, byte data (ECI \000899) with Start Codeword 280.
- Whilst design has been optimized for ECI, normal rules apply:
  - ECI indicators can be directly encoded in the data, for example to allow for multi-culture segmentation of the message.
  - An AIM symbology identifier is mandatory to indicate to the host that ECI message encoding is in effect to avoid ambiguous decoding!

Ultracode Feature: GS1 Digital Link

- Explicit indication that a symbol contains a GS1 Digital Link URI.
  - Akin to “FNC1 in first position” representing GS1 AI element string data.
  - Avoids need for reader / application to resort to a heuristic to guess whether the payload represents a Digital Link URI, or is merely similar.

https://id.gs1.org/**01**/1**23**12312312333**/22/** ... ?3103=000195&3922=0299...

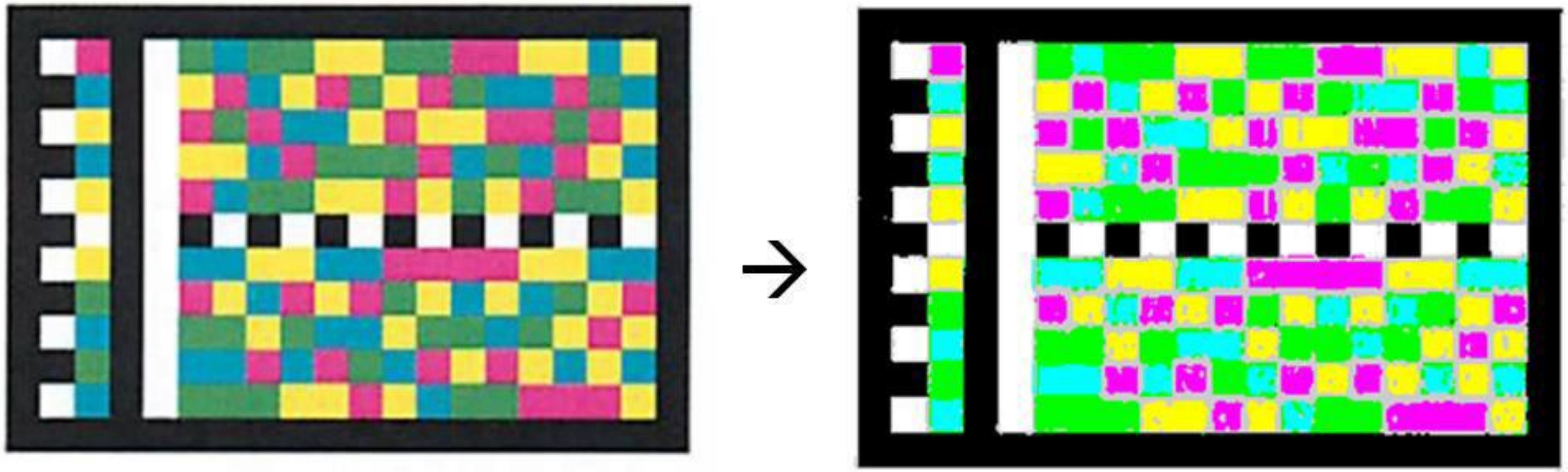
- Start CW [274] encodes https://id.gs1.org/ and sets the AIM symbology identifier modifier to unambiguously indicate a Digital Link URI payload.
- Custom delimiter in double-density numeric data permits efficient encoding of long runs of “/”-separated digits.
- “?”, “&”, “=“, “%” are in a compact character set.

	[279] [47] Enter double-density numeric submode with “/” delimiter
01 /1	[129] [260]
23 12	[151] [140]
…	
3/ 22 **/**	[251] [150] [47]

Ultracode Color Decoding

Ultracode Deals with Real World Lighting

- Electronic displays *generate* light, overriding ambient lighting
- Ultracode's novel method - Does not *measure* colors, it *classifies* them into C, M, Y and G – Highly tolerant of color hue variation
- Scanning example:



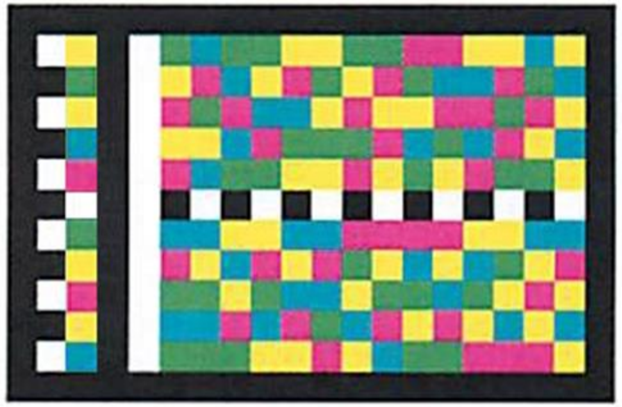
Color decoding in Ultracode

- sRGB digital cameras, as used in smartphones as assumed

Color	Ideal Red	Ideal Green	Ideal Blue
C Cyan	0	255	255
M Magenta	255	0	255
Y Yellow	255	255	0
G Green	0	255	0
K Black	0	0	0
W white	255	255	255

- In reality, especially with printed symbols, both the printing and lighting brightness and color temperature affect measured values

A visually “good” laser printed symbol

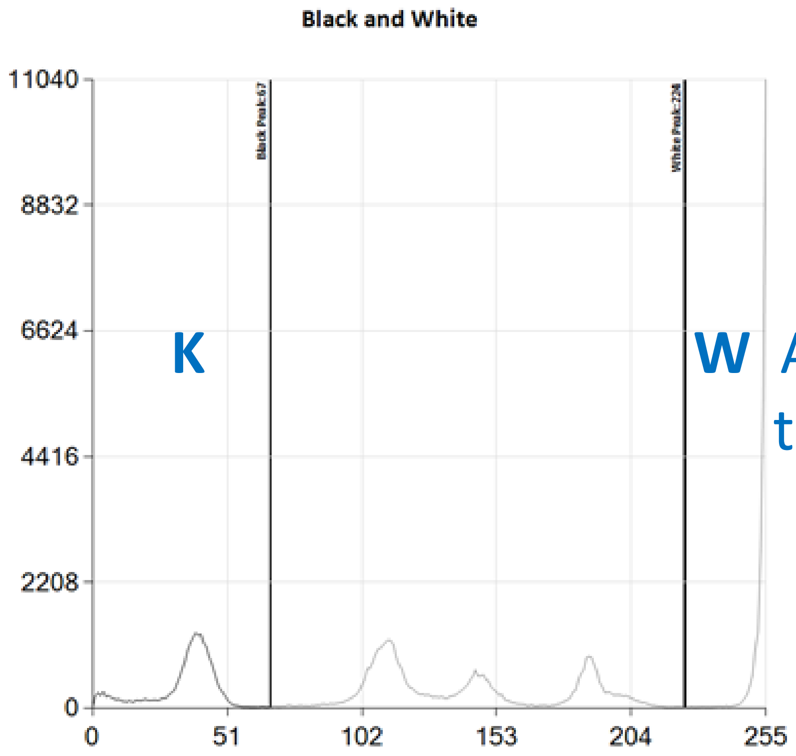


Avg Pixel	Grey	R	G	B	Test Image	Ideal Colour
C	114	1	156	186		
M	143	223	63	142		
Y	190	252	232	85		
G	106	72	150	96		
K	42	39	41	45		
W (paper)	255	255	255	255		

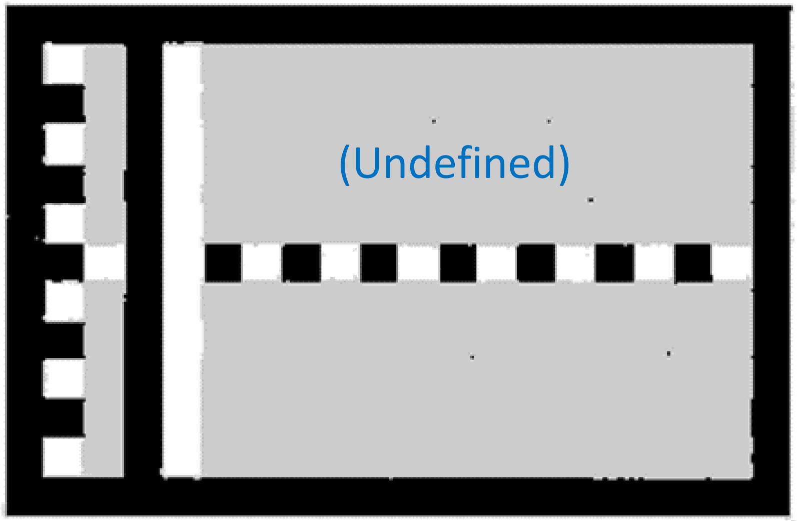
- Black and white are fairly easily separated using grey values
- But black and white modules are only used for symbol structure (frame, horizontal & vertical clock tracks, and finder pattern)
- All the data is contained in the 5 module color tiles
- The real problem is to identify the CMYG colors of the tile modules

Ultracode does not measure colors- It classifies them

- Histogram methods of all bitmap pixels' RGB and grey value used
- Example using grey =(R+G+B)/3 to classify W, K and *undefined*

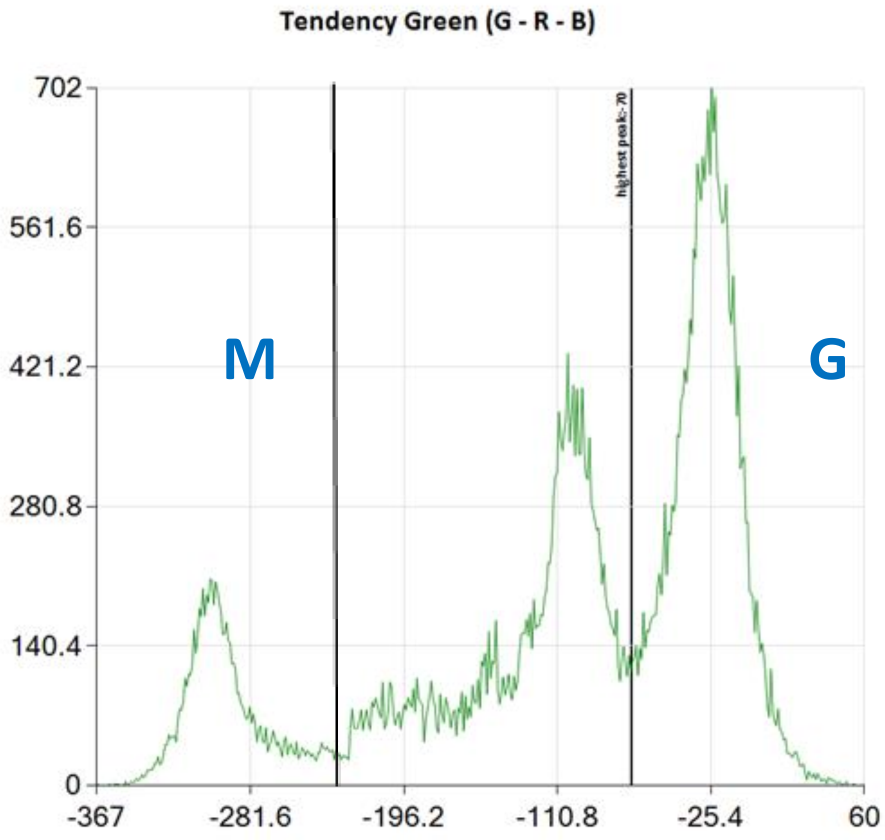


Applied to pixel map to recover structure

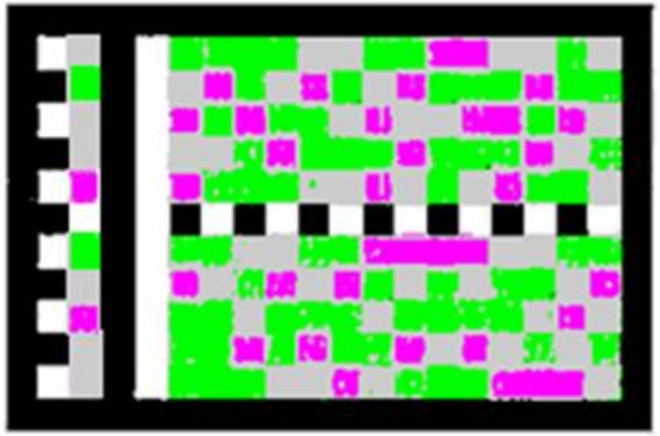


Choice of what you histogram is the key

- Magenta and green are complementary colors in RGB space
- Define Tendency to Green, TG = (G - R - B) to classify M,G colors

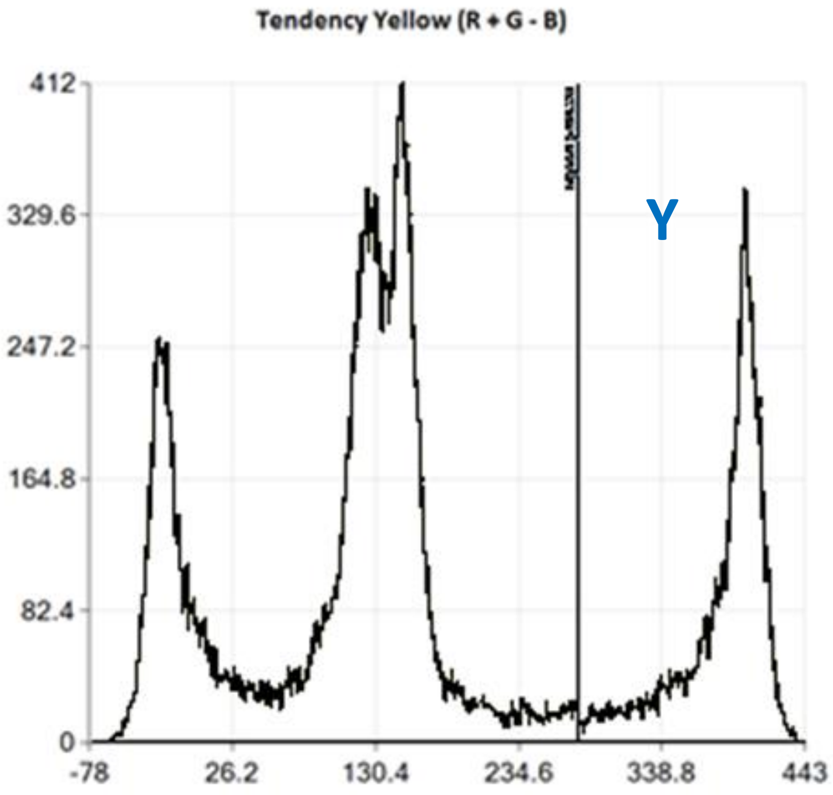


classifies pixels in bit map as:

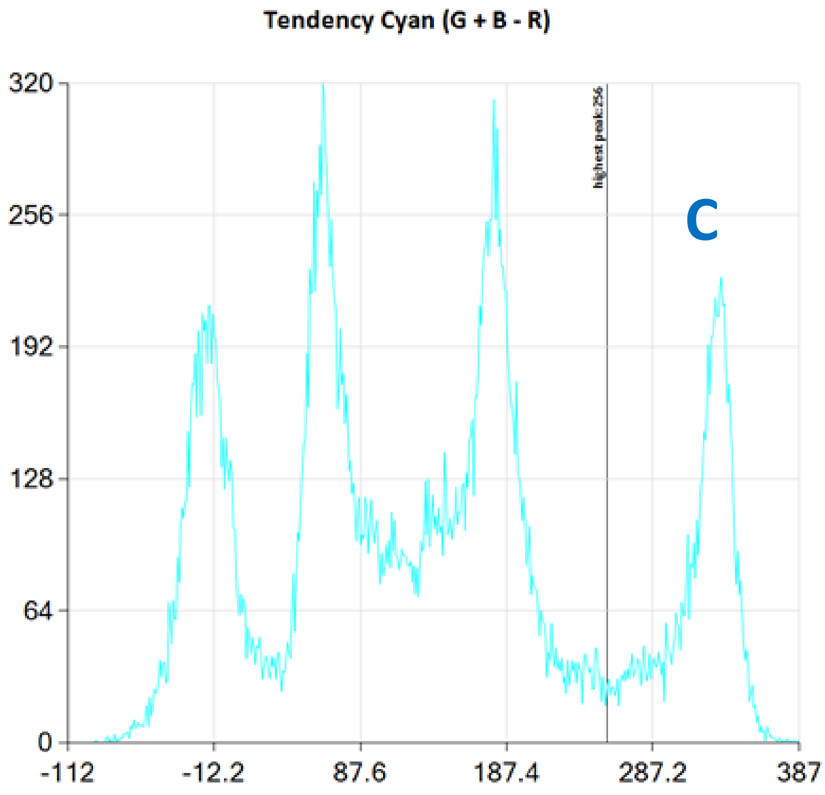


Similarly define tendencies to yellow and cyan

TY = (R + G – B)

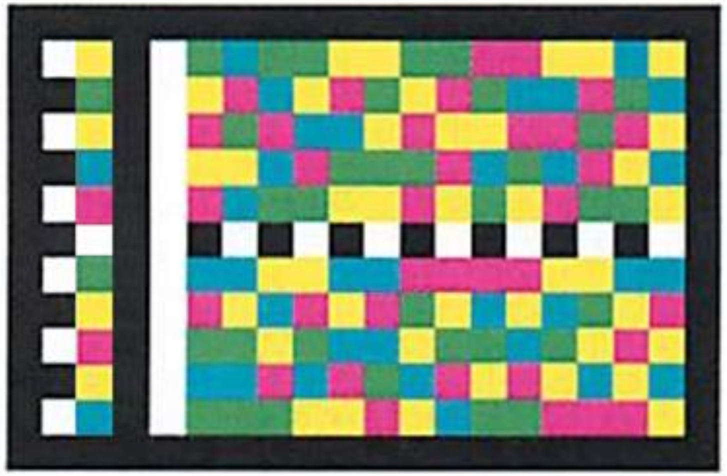


TC = (G + B – R)

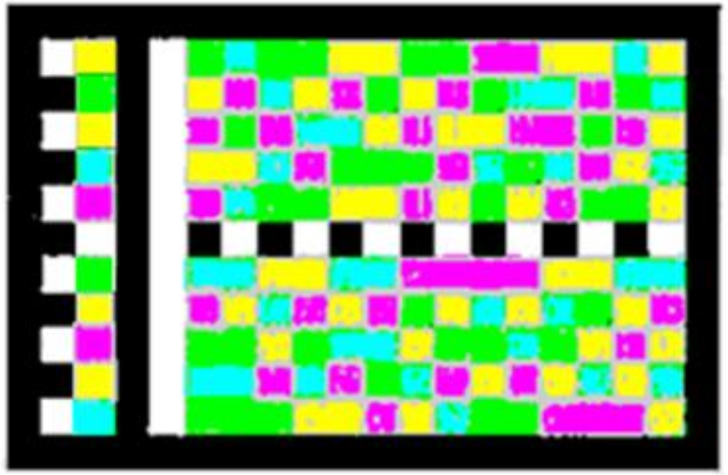


Combining all identified pixel colors

- Laser image pixels are now classified as canonical CMYGKW:



Scanned bitmap identified as:



- Method works because you have a limited number of colors, and clear differentiation between structural colors W,K and codeword tile colors CMYG
- Robust: Works with wide ranges of printing color and lighting

Questions and comments

Thanks for attending!

Dr Clive Hohberger cph13@case.edu
Terry Burton tez@terryburton.co.uk