Real-Time Barcode Recognition

Kunmo Kim and Yiwei Cheng, TAMU
Course Instructor: Professor Deepa Kundur

Introduction

The demonstration is a prototype of 1D barcode scanning using the DSP DM6437EVM board. Barcode encodes data on parallel lines of different widths. The most universally used barcode is the UPC, Universal Product Code. The most common form of the UPC is the UPC-A, which has 12 numerical digits encoded through varying width of black and white parallel lines. The UPC-A barcode is an optical pattern of bars and spaces that format and encode the UPC digit string. Each digit is represented by a unique pattern of two bars and two spaces. The bars and spaces are variable width; they may be 1, 2, 3, or 4 units wide. The total width for a digit is always 7 units. Since there are 12 numbers, the barcode has starting lines, middle separator, and ending lines. A complete UPC-A includes 95 units: the 84 for the left and right digits combined and 11 for the start, middle, and end patterns. The start and end patterns are 3 units wide and use the pattern bar-space-bar; each bar and space is one unit wide. The middle pattern is 5 units wide and uses the pattern space-bar-space-bar-space, also one unit wide. In addition, a UPC symbol requires a quiet zone (additional space) before the start and after the end.The second set of 6 numbers after the middle separator uses the same encoding format of the numerical values of the first 6, except the black and white widths are reversed.

Fig. 1. UPC-A Barcode (“Barcode.”).

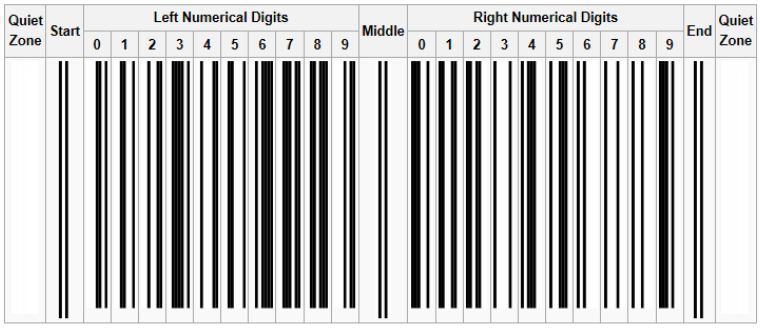

Fig. 2. Number Encoding (“Universal Product Code.”).

The algorithm implemented in this prototype reads the UPC barcode through modules of video input, color conversion, feature calculations, barcode recognition, barcode validation, and output video display.

Color Conversion

Using video capture from the board, the image is taken from the camera to Simulink and is converted from YCrCb to RGB for better processing in Simulink. The conversion requires taking the YCrCb and splitting it into the three color signals of Y, Cr, and Cb. After the split, since the Cr and Cb are smaller in dimension than Y, the Cr and Cb are upsampled using chroma resampling and transposed to match the dimensions of RGB from the 4:2:2 to 4:4:4. The three color signals are transposed again before sending them to the color space conversion from YCrCb to RGB still in three separate signals. The separate RGB signals are concatenated with a matrix concatenate for one to use as display, and for

another line, it is sent to convert from RGB to intensity. The grayscale version of the image will be inserted to the feature calculations.

This process of color conversion is also reversed before sending to output of board, except in this case, it will be from RGB to YCrCb.

Feature Calculations

The feature calculations module of the algorithm creates 3 scanlines for scanning barcodes as well as calculating the pixel values from the barcode intensity image in a given row to a vector. First a Gaussian filter is implemented to smooth out the image gradient identified as the barcode region. The gradient of the scanlines are set and validated so that the scanlines are inside the appropriate range. Then, the mean and standard deviation of the pixel intensities are calculated for the barcode area. The range of pixel parameters, f_low and f_high, for setting the color is determined. Pixels on the scanlines are compared to the f_low and f_high intensity values. A pixel is considered black if its value is less than f_low, and it is considered white if its value is f_high or larger. The remaining pixels are proportionally set between white and black. Black pixels are set to 1 and white pixels are set to -1. From the calculations, the vector of pixels from the scanlines is inputted to the barcode recognition. The scan lines are also sent to display to be added to the real time video.

Barcode Recognition

The barcode recognition module consists of three parts: bar detection, barcode detection, and a barcode comparison block. The bar detection block detects bars from the barcode feature signal. First, it tries to identify a black bar, if it is not there, then the first bar has zero width. If there is a black bar, then it calculates the pixels of the black bar. For the white bars, it does the same. After the bar detections, the barcode detection begins with the beginning bars and calculates all the possible values of barcode values that may form a valid string with all the possible separators. This function returns sequence of indices to barcode guard bars. The barcode comparison block takes in the codebook for all the encoded GTIN 13 barcode values. It also reverses it for determining the last 6 digits of the GTIN 13 barcode. The barcode recognition block takes in the barcodes and tries to match up the barcode with the numbers of pixels generated from the bar detection. In order to ensure better accuracy, the values are calculated from the left to right and right to left. The normalized confidence is calculated. The barcode recognition block set returns the barcode and the normalized confidence.

Barcode Validation

In the barcode validation stage of the algorithm, the simple calculation is used to determine whether the barcode is valid or not. It is calculated by taking the even elements and multiplying them by three. Then, add the sum of the odd elements with the sum of the even elements. Take 10 mod the sum and subtract 10. If the answer is the same as the check digit, which is the last digit, then the barcode is valid. This validation along with a confidence level higher than the threshold allows the barcode to be displayed on the screen.

Display

The display adds the scanlines to the real time video and displays the barcode only if it is validated and has a high enough confidence level to enable the switch for display. All the information is

sent to the module to convert the 3 dimensional matrices back to 2D matrices. Then, RGB is converted to YCrCb format to display through the board.

Design and Implementation

Digital Video Hardware Implementation

1. Make sure the Simulink model works at the system level. Most of the blocks implemented in the system level Simulink simulation will be used in the hardware implementation as well.
2. Go to the Simulation -> Configuration Parameters, change the solver options type to "Fixed-step", solver to "discrete (no continuous states)."
3. Under Real-Time Workshop tab in Configuration Parameters, change system target file' to ccslink_grt.tlc. Then, go to "Embedded IDE Link" tab, and set System stack size (MAUs) to 8192. Click OK and exit the Configuration Parameters.
4. Click Library Browser. Under Target Support Package→Supported Processors→TI C6000→Board Support→DM6437EVM, find Video Capture block. Set Sample Time to -1.
5. Add DM6437 Video Capture block, DM6437 Video Display block, and DM6437EVM block. These blocks are your input, output, and target preferences, respectively. Double click the Video Capture block, and set Sample Time to -1, Video capture mode to NTSC, and Analog video input to composite. Double click the Video Display, and set Video window to Video 0, Video window position to [0, 0, 720, 480], Horizontal zoom to 1x, and Vertical zoom to 1x. Click OK.
6. Connect Video Capture to Deinterleave block and Interleave block to the Video Display. Since other features, including scan lines and barcode numbers, will be added on top of the original video, Deinterleave and Interleave blocks should be employed to process data in a certain sequence. Otherwise, it will generate an error during the cross-compiling (Fig. 3.).
7. Transpose the output Cb and Cr of the Deinterleave block, and link them to the Chroma Resampling block. Double click the Chroma Resampling block, and set the Resampling to "4:2:2 to 4:4:4" (Fig. 4.).
8. Transpose the output Y of the Deinterleave as well.
9. Add a Color Space Conversion block, and set Conversion to "Y'CbCr to R'G'B'", Image signal to Separate color signals. Connect those transposed data to the Color Space Conversion block.
10. Add Matrix Concatenate block and set both of Number of inputs and Concatenate Dimension to 3. Connect the output of Color Space Conversion block to the Matrix Concatenate block.
11. Double click the R'G'B' to Intensity Color Space Conversion block. Image signal should be set to Separate color signals.
12. Double click the Row Positions Of Scanlines block, and set Constant value to [200 240 280]. These values will be used for the position of scan-lines. [200 240 280] will draw three lines in the center of the screen.
13. Create a Barcode Recognition block by using the M-code attached in Appendix. The configuration of the Barcode Recognition block is displayed in Fig. 6. **All the codes used in this project are borrowed from Matlab – Barcode Recognition example.**
14. Create a Barcode Validation block. The configuration is displayed in Fig. 7. This configuration is just a transformed equation into block-level on Simulink. The equation exploits $T=2i=05d2i+i=010di$ and $C=10-(T \bmod 10)$, where di is the i-th digit in the barcode with d0 as the first digit, and C is the check bit. The equations are originated from the paper "Fundamentals of Barcode Information Theory" (Pavlidis, Swartz, and Wang).

15. Go to the inside of the Display block. Double click the Draw Shapes block, and change the Color value to [0 255 0]. Now, the color of scan lines is set to green. (Fig. 8.)
16. In the Display block, create three selector blocks and put them after the switch. Set Number of input dimentions to 3, first and second indexloption to select all, and third indexloption to Index vector (dialog) for three selector blocks. Assign 1 to the third indexloption of the first selector block. Second selector block has 2 and and third selector block has 3 for this option (Fig. 5.).
17. Create a Color Space Conversion block. Double click this block, then set Conversion to "R'G'B' to Y'CbCr" and Image signals to Separate color signals. Transpose all the three outputs.
18. Create a Chroma Resampling block, and set Resampling to 4:4:4 to 4:2:2. Connect transposed Cb and Cr signals to the Chroma Resampling block.
19. Connect Transposed Y' signal, and resampled Cb and Cr signals to Interleave block. Then, connect output of the Interleave block to the Video Display block.
20. Connect the video output (the yellow port) of the camera to the video input of the board, and the output of the board to the monitor.
21. Connect the video output port on the DSP board to the Video 0 input located back of the TV in the lab.
22. Go to Tools -> Real Time Workshop -> Build the model.

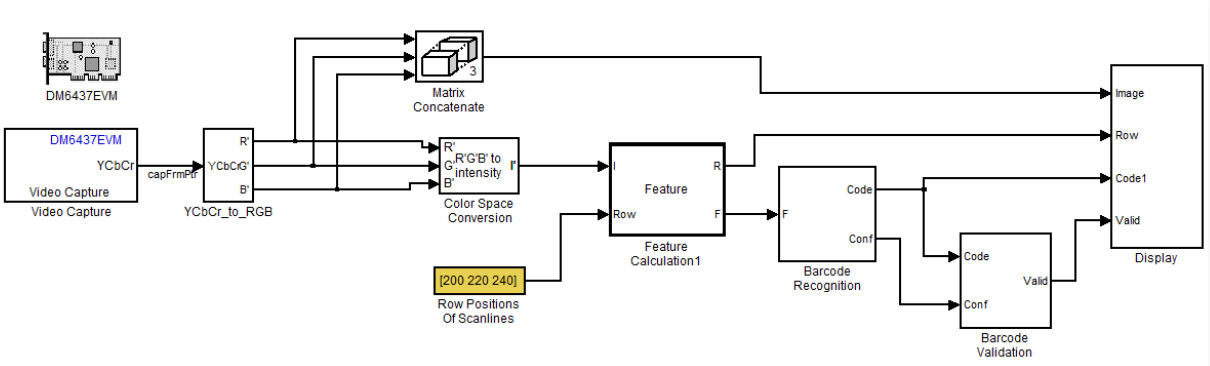

Fig. 3. Barcode Recognition Simulink Model Implemented on DM6437 Board.

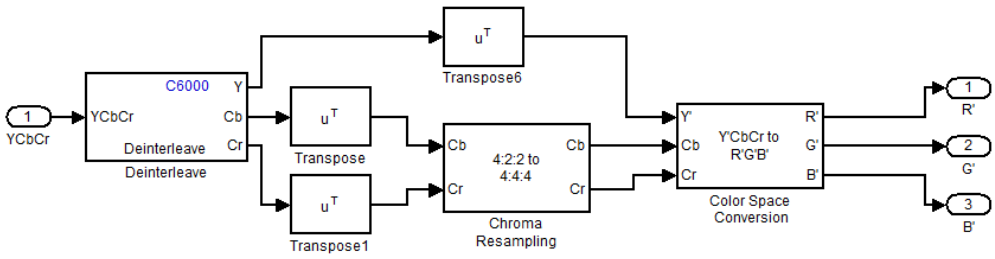

Fig. 4. Conversion from YCbCr to RGB.

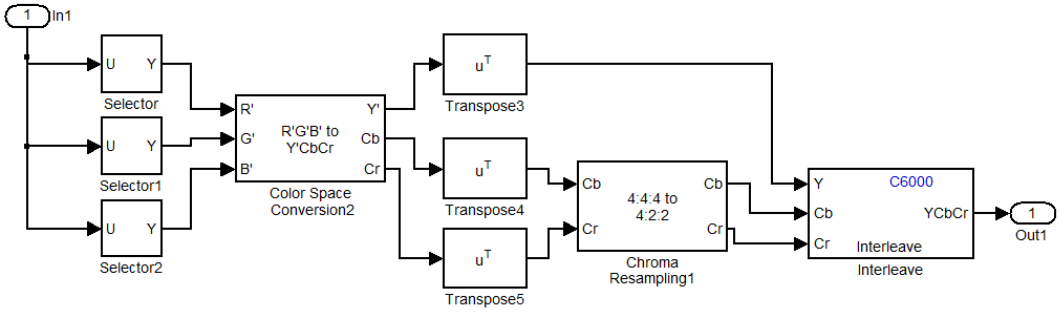

Fig. 5. Conversion from RGB to YcBCr for Video Display of DM6437 Processor.

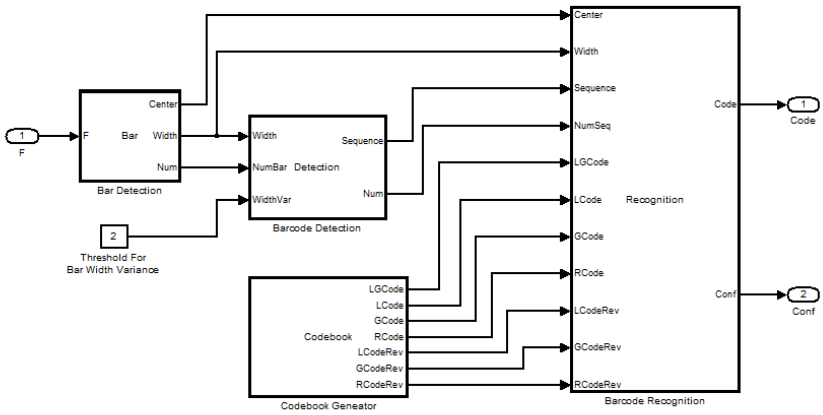

Fig. 6. Barcode Recognition Block.

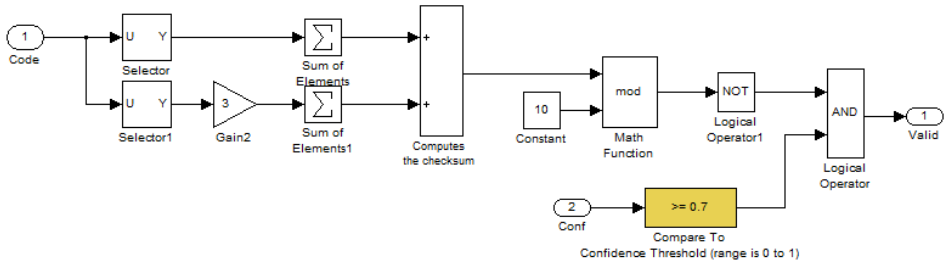

Fig. 7. Barcode Validation Block.

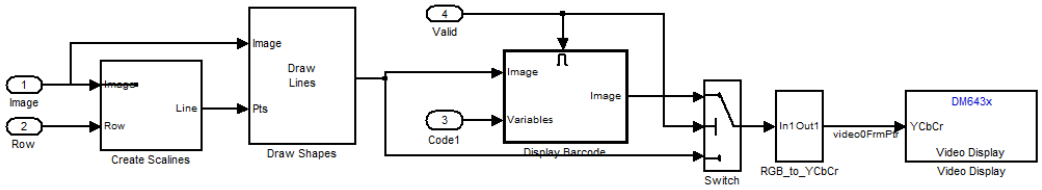

Fig. 8. Display Block.

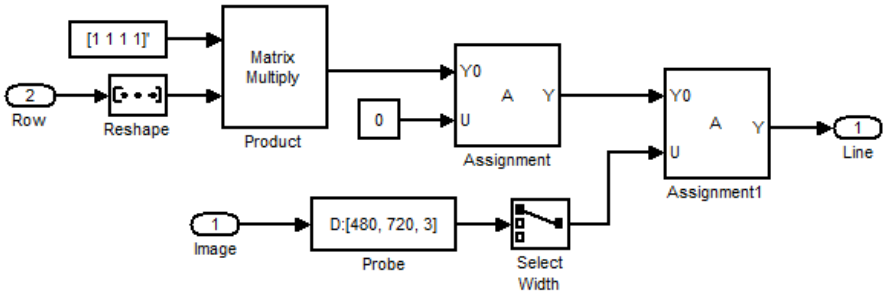

Fig. 9. Create Scanelines Block (located inside of Display block).

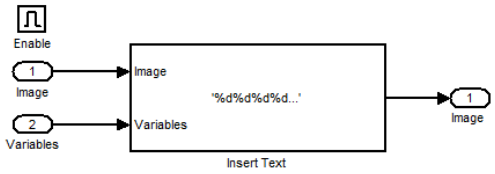

Fig. 10. Display Barcode Block (located inside of Display block).

Feature Calculation (Appendix A.)

Feature Calculation simply transforms input pixels into a vector. The output R directly produces the number from input Row. This output will be used in Display block later again. Once all the pixels within each scan-line are transformed vectors, then block reads the pixel information. It uses a Gaussian filter to smooth the image gradient to improve the identification of barcode region. After the validation of pixels in scan-lines, the block calculates the mean and standard deviation values. Block uses this information to compute the highest and lowest intensity. Once the highest and lowest intensities are distinguished, which should be black and white from now on, the block calculates which pixels are considered in black color and which are in white color. The pixel with intensity higher than the highest intensity will be set to -1, and intensity lower than the lowest intensity will be set to 1. Other than these will be set to a value between -1 and 1, which will be discarded later.

Bar Detection (Appendix B.)

Using the information received from the Feature Calculation block, it simply recognizes the black and white colors, and their width, numbers, and the position of the center of black and white pixel bars.

Barcode Detection (Appendix C.)

The barcode detection detects the indices of three barcode guard bars obtained from Bar Detection block. The UPC-A barcode type includes a total of 56 black and white bars. From this information, the block can recognizes and validates the start and end point of guard bars as well as the position of barcodes. The block firstly tries to find out all possible separators by recognizing the black and white bar pattern. Once all the separators are detected, then sequence of bars can be obtained through somewhat complex calculation. The detailed computation algorithms are described in the paper “Fundamentals of Barcode Information Theory” (Pavlidis, Swartz, and Wang).

Codebook Generator (Appendix D.)

The codebook is originated from “Fundamentals of Barcode Information Theory” (Pavlidis, Swartz, and Wang).

Barcode Recognition

The block finally produces the actual number decoded from the barcode captured through the input camera. Since we detected the guard bars before, this information also can be used to identify if the sequence of barcodes are flipped over. If the barcodes are flipped over, then the block reads the barcode from right to left. Therefore, the bar codes can be detected in any case unless it is rotated by 90 degrees. To employ this function, we may need three more vertical scan-lines and check whether vertical or horizontal lines contain barcode information. Once the direction of the barcode is identified, then the block finds the potential match for digits between 2 and 7 for the first half of the barcode. The barcode will be decoded through the codebook used in Appendix D. Then, the second half of the barcode, which contains 6 digits, will be decoded. These two decoded numbers will be concatenated and assigned to a variable ‘code’. The ‘code’ variable will be sent to barcode validation blocks, and then will be displayed on the monitor if it is confirmed as valid numbers.

Results

The barcode recognition program is successfully implemented on Simulink and cross-compiled to the DM6437 DSP processor. The Simulink model of sub-blocks of Barcode Recognition, consisting of YCbCr_to_RGB block, Barcode Recognition block, Barcode Validation block, and Display are illustrated in Figs 3, 4, 5, 6, 7, 8, 9 and 10. These blocks are used in the hardware implementation with DM6437 Evaluation Module (EVM) DSP board provided in the lab.

The hardware setup is displayed in Figs. 11, 12, 13, 14, and 15. Fig. 1 shows the DM6437 EVM DSP board. +5V power is supplied from the cable connected at bottom, and USB for data transmission is provided at top of the board. On the left side of the board (Fig. 12), the camera (Fig. 13) is connected to the Video Input port, and the Video Output port is connected to the back panel of the TV (Figs. 14 and 15).

After the program is successfully cross-compiled and loaded on the DSP processor, TV displays the screen captured through a camera in real time, and DSP processor prints out a number corresponding to a scanned barcode as shown in Fig. 16.

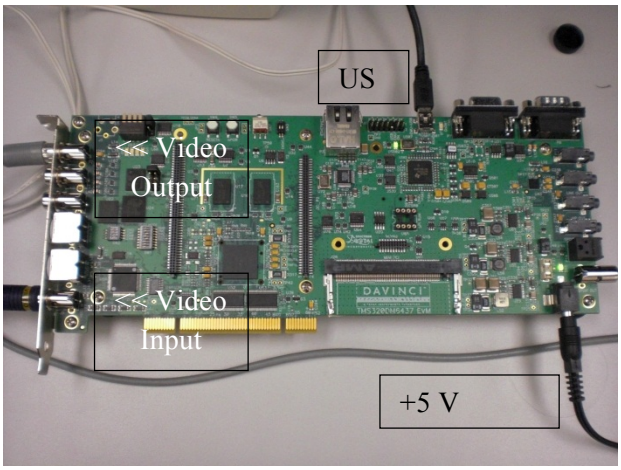

Fig. 11. DM6437 DSP Evaluation Module.

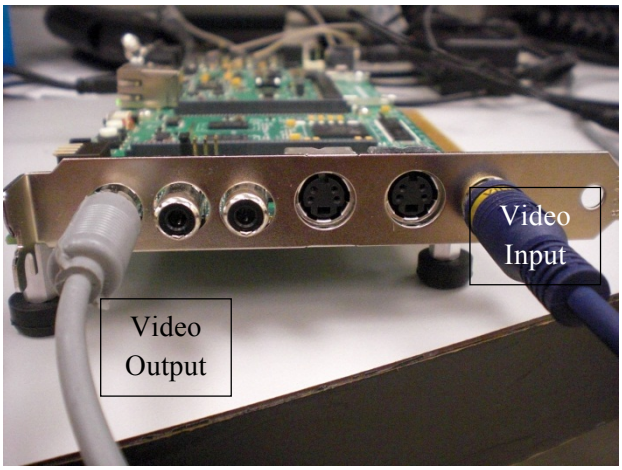

Fig. 12. Video input (right) and output (left).

Fig. 13. Camera (Video Input).

Fig. 14. TV (Video Output).

Fig. 15. Connection of Video Output and TV.

Fig. 16. Screen Shot of Barcode Recognition Program.

References

"Barcode." Wikipedia, The Free Encyclopedia. Wikimedia Foundation, Inc. 8. Dec. 2011. Web. 9. Dec. 2011.

"Universal Product Code." Wikipedia, The Free Encyclopedia. Wikimedia Foundation, Inc. 2 Dec. 2011. Web. 2 Dec. 2011.

Pavlidis, Theo, Jerome Swartz, and Ynjiun P. Wang. "Fundamentals of bar code information theory." *Computer* 23.4 (1990): 74-86. Print.

Appendix

A. Matlab code for Feature Calculation

```
function [R, F] = Feature(I, Row)
% This function transforms pixels in a given row into a feature vector.

RATIO_STD = single(0.4); % Parameter

% h is a Gaussian filter. It is used to smooth the image gradient which
% indentifies barcode region.
h = single([0.022191, 0.045589, 0.079811, 0.119065, 0.151361, 0.163967, ...
            0.151361, 0.119065, 0.079811, 0.045589, 0.022191]);

% Make sure all scanlines are in valid range.
R = Row;
numRow = size(I, 1);
for idx = 1: numel(R)
    if R(idx) < 1
        R(idx) = 1;
    elseif R(idx) >= numRow
        R(idx) = numRow;
    end
end

F = single(I(R, :));
len = numel(F);

% gradient of the scanlines
gradient = zeros(size(F), 'single');
gradient(2:end-1) = abs(F(3:end) - F(1:end-2));
gradient = conv2(gradient, h, 'same');

% mask consists of most pixels in the barcode region.
mask = (gradient > mean(mean(gradient)));
numPix = single(sum(sum(mask)));
f_mean = single(sum(sum(F .* mask))) / numPix;

% Calculate the mean value and standard deviation for the pixels in the
% barcode region.
f_std = zeros(1, 'single');
for idx = 1: len
    if mask(idx)
        dif = F(idx) - f_mean;
        f_std = f_std + dif * dif;
    end
end
f_std = sqrt(f_std / numPix);

% Estimate the range of pixel intensity values in the barcode region.
% Values larger than f_high and values smaller than f_low will be saturated.
f_high = f_mean + f_std * RATIO_STD;
f_low  = f_mean - f_std * RATIO_STD;

% Calculate the feature for all pixels in the scanlines. A pixel is
% classified as black if its value is f_low or less. Black pixels are set
% to 1. A pixel is classified as white if its value is f_high or larger.
% White pixels are set to -1. The remaining pixels are proportionally set
% to values between -1 and 1.
scale = single(2) / (f_high - f_low);
for iPix = 1: len
    if F(iPix) > f_low && F(iPix) < f_high
        F(iPix) = (f_high - F(iPix)) * scale - single(1);
    elseif F(iPix) <= f_low
        F(iPix) = single(1);
    else
        F(iPix) = single(-1);
    end
end

% Conver the coordinates of from 1-based to 0-based.
R = R - 1;
```

B. Bar Detection

```
function [Center, Width, Num] = Bar(F)
% Function BAR detects bars from barcode feature signal.

MAX_BAR_NUM = 200;
[numFeature, lenFeature] = size(F);
Center = zeros([MAX_BAR_NUM, numFeature], 'single');
Width = zeros([MAX_BAR_NUM, numFeature], 'single');
Num = zeros([1, numFeature], 'int32');

for iFeature = 1: numFeature
    iBar = 0;
    iPix = 1;

    % First, try to find a black bar.
    % If it is not there, the first bar has zero width.
    while iPix <= lenFeature && iBar < MAX_BAR_NUM
        % Find a black bar.
        % A contiguous sequence of pixels with zero or positive feature
        % value is considered a black bar.
        curWidth  = zeros(1, 'single');
        curCenter = zeros(1, 'single');
        while iPix <= lenFeature && F(iFeature, iPix) >= 0
            curWidth  = curWidth + F(iFeature, iPix);
            curCenter = curCenter + F(iFeature, iPix) * iPix;
            iPix = iPix + 1;
        end

        iBar = iBar + 1;
        if curWidth > 0
            Center(iBar, iFeature) = curCenter / curWidth;
            Width(iBar, iFeature) = curWidth;
        end

        % Find a white bar.
        % A contiguous sequence of pixels with negative feature
        % value is considered a white bar.
        curWidth  = zeros(1, 'single');
        curCenter = zeros(1, 'single');
        while iPix <= lenFeature && F(iFeature, iPix) < 0
            curWidth  = curWidth + F(iFeature, iPix);
            curCenter = curCenter + F(iFeature, iPix) * iPix;
            iPix = iPix + 1;
        end

        if curWidth < 0 && iBar < MAX_BAR_NUM
            iBar = iBar + 1;
            Center(iBar, iFeature) = curCenter / curWidth;
            Width(iBar, iFeature) = -curWidth;
        end
    end
    Num(1, iFeature) = iBar;
end
```

C. Barcode Detection

```
function [Sequence, Num] = Detection(Width, NumBar, WidthVar)
% DETECTION returns sequence of indices to barcode guard bars

% Parameters
WIDTH_VAR_DETECT = WidthVar;
BAR_VAR_NUM = 1;

% Maximum signal dimensions
MAX_SEP_NUM = 50; % max number of guard patterns (separators)
MAX_SEQ_NUM = 30; % max number of sequences of guard patterns

% Constant values
BAR_END2END_NUM = 56; % count of both black and white bars
BAR_END2MID_NUM = 28;

numScanline = numel(NumBar);
separator = zeros([1, MAX_SEP_NUM], 'int32');
Sequence = zeros([3*MAX_SEQ_NUM, numScanline], 'int32');
Num = zeros([1, numScanline], 'int32');

for iScanline = 1: numScanline
    % Calculate sequence and its length
    firstBar = 1;
    % Skip the first bar if it's invalid
    if Width(firstBar) == 0
        firstBar = firstBar + 2;
    end

    % Find out all possible separators
    numSep = 0;
    while firstBar <= NumBar(iScanline)-2
        [avgWidth, devWidth] = Statistics(Width, iScanline, firstBar, 3);
        if numSep < MAX_SEP_NUM && devWidth < WIDTH_VAR_DETECT; % * dis_mean
            numSep = numSep + 1;
            separator(numSep) = firstBar;
        end
        firstBar = firstBar + 2;
    end

    % Find out sequences of bars that may form a valid barcode string
    iSequence = 0;
    iStart = 1;
    lastStartSep = NumBar(iScanline) - BAR_END2END_NUM + 2 * BAR_VAR_NUM;
    while iSequence < MAX_SEQ_NUM ...
            && iStart <= numSep-2 && separator(iStart) <= lastStartSep
        firstMidSep = separator(iStart) + BAR_END2MID_NUM - 2 * BAR_VAR_NUM;
        lastMidSep  = firstMidSep + 4 * BAR_VAR_NUM;
        firstEndSep = separator(iStart) + BAR_END2END_NUM - 2 * BAR_VAR_NUM;
        lastEndSep  = firstEndSep + 4 * BAR_VAR_NUM;

        % Skip the first part
        iMid = iStart + 1;
        while iMid <= numSep-1 && separator(iMid) < firstMidSep
            iMid = iMid + 1;
        end

        while iSequence < MAX_SEQ_NUM ...
                && iMid <= numSep-1 && separator(iMid) <= lastMidSep
            % Skip the first part
            iEnd = iMid + 1;
            while iEnd <= numSep && separator(iEnd) < firstEndSep
                iEnd = iEnd + 1;
            end

            while iSequence < MAX_SEQ_NUM ...
                    && iEnd <= numSep && separator(iEnd) <= lastEndSep
                avgWidth = zeros(1, 'single');
                for iBar = separator(iStart): separator(iStart)+2
                    avgWidth = avgWidth + Width(iBar, iScanline);
                end
                for iBar = separator(iEnd): separator(iEnd)+2
                    avgWidth = avgWidth + Width(iBar, iScanline);
                end
                avgWidth = avgWidth / single(6);

                devWidth = zeros(1, 'single');
```

```
                for iBar = separator(iMid)-1: separator(iMid)+3
                    difWidth = abs(Width(iBar, iScanline) - avgWidth);
                    if devWidth < difWidth
                        devWidth = difWidth;
                    end
                end

                if devWidth < WIDTH_VAR_DETECT; % * dis_mean_exp
                    Sequence(3*iSequence+1, iScanline) = separator(iStart);
                    Sequence(3*iSequence+2, iScanline) = separator(iMid);
                    Sequence(3*iSequence+3, iScanline) = separator(iEnd);
                    iSequence = iSequence + 1;
                end
                iEnd = iEnd + 1;
            end
            iMid = iMid + 1;
        end

        iStart = iStart + 1;
    end
    Num(iScanline) = iSequence;
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the average and deviation of the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [avg, dev] = Statistics(data, col, first, Num)
avg = zeros(1, 'single');
for idx = first: first+Num-1
    avg = avg + data(idx, col);
end
avg = avg / Num;

dev = zeros(1, 'single');
for idx = first : first+Num-1
    dif = abs(data(idx, col) - avg);
    if dev < dif
        dev = dif;
    end
end
end
```

D. Codebook Generator

```
function [LGCode, LCode, GCode, RCode, LCodeRev, GCodeRev, RCodeRev] = Codebook
UPSAMPLE_RATE = int32(16); % Parameter
% Function CODEBOOK generats the codebooks for GTIN-13.

%------------------------------------------------------------------------
% Codebook for the first digit which is encoded in the L/G patterns.
%------------------------------------------------------------------------
ean_code_sym = [ ...
    'LLLLLL'; ...
    'LLGLGG'; ...
    'LLGGLG'; ...
    'LLGGGL'; ...
    'LGLLGG'; ...
    'LGGLLG'; ...
    'LGGGLL'; ...
    'LGLGLG'; ...
    'LGLGGL'; ...
    'LGGLGL' ];

LGCode = zeros([10, 6], 'int32');
for iNum = 1: 10
    for iSym = 1: 6
        if ean_code_sym(iNum, iSym) == 'L'
            LGCode(iNum, iSym) = iSym;
        else
            LGCode(iNum, iSym) = iSym + 6;
        end
    end
end

%------------------------------------------------------------------------
% Codebook for the other 12 digits
%------------------------------------------------------------------------
lcode_sym = [ ...
    '0001101'; ...
    '0011001'; ...
    '0010011'; ...
    '0111101'; ...
    '0100011'; ...
    '0110001'; ...
    '0101111'; ...
    '0111011'; ...
    '0110111'; ...
    '0001011' ];

gcode_sym = [ ...
    '0100111'; ...
    '0110011'; ...
    '0011011'; ...
    '0100001'; ...
    '0011101'; ...
    '0111001'; ...
    '0000101'; ...
    '0010001'; ...
    '0001001'; ...
    '0010111' ];

rcode_sym = [ ...
    '1110010'; ...
    '1100110'; ...
    '1101100'; ...
    '1000010'; ...
    '1011100'; ...
    '1001110'; ...
    '1010000'; ...
    '1000100'; ...
    '1001000'; ...
    '1110100' ];

[LCode] = convertAndUpsample(lcode_sym, UPSAMPLE_RATE);
[GCode] = convertAndUpsample(gcode_sym, UPSAMPLE_RATE);
[RCode] = convertAndUpsample(rcode_sym, UPSAMPLE_RATE);

[LCodeRev] = reverseCodebook(LCode);
[GCodeRev] = reverseCodebook(GCode);
[RCodeRev] = reverseCodebook(RCode);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert the codebook from symbols to digits and upsample them.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [codeExpand] = convertAndUpsample(code_sym, UPSAMPLE_RATE)

numCode = size(code_sym, 1);
lenCode = size(code_sym, 2);
codeExpand = zeros([numCode, lenCode*UPSAMPLE_RATE], 'int32');

for iCode = 1: numCode
    for bit = 1: lenCode
        if code_sym(iCode, bit) == '0'
            val = int32(-1);
        else
            val = int32(1);
        end

        for idx = (bit-1)*UPSAMPLE_RATE+1: bit*UPSAMPLE_RATE
            codeExpand(iCode, idx) = val;
        end
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reverse the codebook
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [codeRev] = reverseCodebook(code)
codeRev = code;
lenCode = size(code, 2);
for idx = 1: lenCode
    codeRev(:, idx) = code(:, lenCode-idx+1);
end
end
```

E. Barcode Recognition

```
function [Code, Conf] = Recognition(Center, Width, ...
    Sequence, NumSeq, LGCode, LCode, GCode, RCode, ...
    LCodeRev, GCodeRev, RCodeRev)
% Function RECOGNITION recognizes the barcode.

% Parameters
UPSAMPLE_RATE = int32(16);

% Constant values
% MID_MODULE is the distance between the first '101' pattern and the middle
% '101' pattern.
MID_MODULE  = int32(46);
% LAST_MODULE is the distance between the first '101' pattern and the last
% '101' pattern.
LAST_MODULE = int32(92);
% numBinInSym is the number of bins in a symbol
numBinInSym = int32(7 * UPSAMPLE_RATE);

numScanline = numel(NumSeq);
maxNumBar = size(Center, 1);
mapCenter = single(zeros(1, maxNumBar));
mapWidth = single(zeros(1, maxNumBar));
barBin = zeros([1, LAST_MODULE*UPSAMPLE_RATE], 'int32');

% Recognize the strings
bestConf = int32(0);
bestCode = zeros([13, 1], 'int32');
codeAll = zeros([13, 1], 'int32');

Center = Center / single(LAST_MODULE);
Width = Width / single(LAST_MODULE);
expCenter = single([0; MID_MODULE; LAST_MODULE]);

for iScanline = 1: numScanline
    for iSequence = 0: NumSeq(iScanline)-1
        dataMx = zeros([3, 3], 'single');
        for iSep = 1: 3
            iBar = Sequence(3*iSequence+iSep, iScanline);
            %loc = mean(bar(1, startBar: startBar+2));
            avgCenter = (Center(iBar, iScanline) ...
                + Center(iBar+1, iScanline) ...
                + Center(iBar+2, iScanline)) / single(3);
            dataMx(iSep, :) = [avgCenter^2, avgCenter, 1];
        end

        coeff = inv(dataMx) * expCenter;

        for iBar = Sequence(3*iSequence+1, iScanline) + 3 ...
                 : Sequence(3*iSequence+3, iScanline) - 1
            mapCenter(iBar) = coeff' * ...
                [Center(iBar, iScanline)^2; Center(iBar, iScanline); 1];
            mapWidth(iBar)  = coeff(1:2)' ...
                * [2*Width(iBar, iScanline); 1] * Width(iBar, iScanline);
        end

        val = int32(-1);
        for iBar = Sequence(3*iSequence+1, iScanline) + 3 ...
                 : Sequence(3*iSequence+3, iScanline) - 1
            left = (mapCenter(iBar) - mapWidth(iBar) / 2) ...
                   * single(UPSAMPLE_RATE);
            right = left + mapWidth(iBar) * single(UPSAMPLE_RATE);
            left = max(left, 1);
            right = min(right, single(LAST_MODULE*UPSAMPLE_RATE));
            for idx = round(left): round(right)
                barBin(idx) = val;
            end
            val = -val;
        end

        leftBin = int32(round(1.5 * UPSAMPLE_RATE));
        rightBin = int32(round(48.5 * UPSAMPLE_RATE));

        % Assuming left to right scanline order, recognize digits
        % 8 to 13.
        [rcodeFwd, rconfFwd] = RecognizeHalfCode(RCode, barBin, ...
            rightBin, numBinInSym, false);
```

```
        % Assuming right to left scanline order, recognize digits
        % 8 to 13.
        [rcodeRev, rconfRev] = RecognizeHalfCode(RCodeRev, ...
            barBin, leftBin, numBinInSym, true);

        % Use the scanline direction (left to right or right to left),
        % find out the potential match for digits 2 to 7.
        if sum(rconfFwd) > sum(rconfRev)
            [lcodeRec, lconfRec] = RecognizeHalfCode(LCode, barBin, ...
                leftBin, numBinInSym, false);
            [gcodeRec, gconfRec] = RecognizeHalfCode(GCode, barBin, ...
                leftBin, numBinInSym, false);
            codeAll(8:13) = rcodeFwd;
            confAll = int32(sum(rconfFwd));
        else
            [lcodeRec, lconfRec] = RecognizeHalfCode(LCodeRev, barBin, ...
                rightBin, numBinInSym, true);
            [gcodeRec, gconfRec] = RecognizeHalfCode(GCodeRev, barBin, ...
                rightBin, numBinInSym, true);
            codeAll(8:13) = rcodeRev;
            confAll = int32(sum(rconfRev));
        end

        % Recognize digits 1 to 7.
        [codeLeft, confLeft] = RecognizeFirstHalfCode(LGCode, ...
            lcodeRec, lconfRec, gcodeRec, gconfRec);
        codeAll(1:7) = codeLeft;
        confAll = confAll + confLeft(1);

        if bestConf < confAll
            bestCode = codeAll;
            bestConf = confAll;
        end
    end
end

% Normalize the confidence to [0 1].
Conf = single(bestConf) / single(12*7*UPSAMPLE_RATE);
Code = bestCode;
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recognize the first 7 digits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [code, conf] = ...
    RecognizeFirstHalfCode(LGCode, lcode, lconf, gcode, gconf)
codeComb = [lcode, gcode];
confComb = [lconf, gconf];
code = zeros([7, 1], 'int32');
conf = zeros([7, 1], 'int32');

bestCode = 1;
bestSum = sum(confComb(LGCode(bestCode, :)));
for idx = 2: 10
    curSum = sum(confComb(LGCode(idx, :)));
    if bestSum < curSum
        bestCode = idx;
        bestSum = curSum;
    end
end

code(1) = bestCode - int32(1);
conf(1) = bestSum;
code(2:7) = codeComb(LGCode(bestCode, :));
conf(2:7) = confComb(LGCode(bestCode, :));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recognize half of the barcode by comparing them with a codebook
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [code, conf] = RecognizeHalfCode(codebook, barBin, ...
    firstBin, numBinInSym, isReverse)
code = zeros([6, 1], 'int32');
conf = zeros([6, 1], 'int32');
```

```
iSym = 1;
step = 1;
if isReverse
    iSym = 6;
    step = -1;
end

startBin = firstBin;
for idx = 1: 6
    bestCode = int32(0);
    bestSum = -2*numBinInSym;
    for iCode = int32(1): int32(10)
        sum = int32(0);
        for jdx = 1: numBinInSym
            sum = sum + barBin(startBin+jdx-1) * codebook(iCode, jdx);
        end

        if bestSum < sum
            bestSum = sum;
            bestCode = iCode;
        end
    end
    code(iSym) = bestCode - int32(1);
    conf(iSym) = bestSum;
    iSym = iSym + step;
    startBin = startBin + numBinInSym;
end
end
```