Demo: Real-Time Video Coin Counting

Xiaojie Cai and Mengchao Zhong, TAMU
Course Instructor: Professor Deepa Kundur

Introduction

The Internet has dramatically changed the way we acquire and consume information. We observe this especially with image and video processing. Today, the direct processing of visual signals is faster than ever. Digital image processing allows for the use of complex algorithms that can offer more sophisticated functionality and implementation. For example, it is commonplace to apply image processing for Classification, Feature extraction, Pattern recognition, Projection, Multi-scale signal analysis. Some techniques which are used in digital image processing include: Pixelization, Linear filtering, Principal components analysis, Independent component analysis, Hidden Markov models, Anisotropic diffusion, Partial differential equations, Self-organizing maps, Neural networks, Wavelets.

For video, it is sometimes possible to consider the content simply as a series of ordered still images and then apply image processing frame-by-frame thus extending image processing functionality to the video domain. The same algorithms and principles may apply such as in the case of computer morphology algorithms, color space conversion, etc.

In this project, a real-time coin counting demo based on video and image processing is implemented. The objective of this demo is to count the number of coins such as quarters we used in the laundry room in a relatively fast speed. What's more, we can develop this demo further to accomplish more complicated counting tasks.

Thresholding

Thresholding is the transformation from grayscale image to binary image. Suppose we have an image where there is an object and added with Gaussian noise as in Fig. 1, and its histogram is shown in Fig. 2.

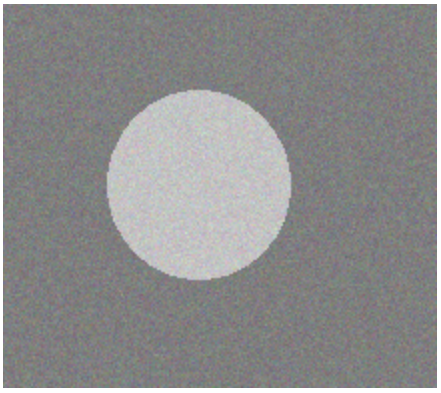


Figure 1 An image with an object in the middle with added Gaussian noise

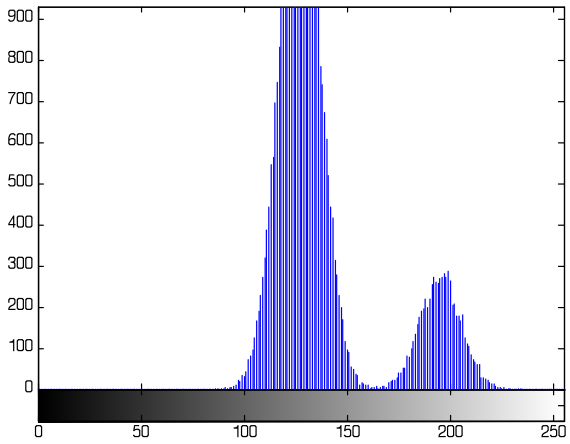


Figure 2 The histogram of figure 1

Thresholding finds an optimum value T, such that pixels with values which are less than T are set to be 0, and pixels with values greater than T are set to 1. The algorithm which is used in the block auto-thresholding is so called Otsu's algorithm. In Rafael C. Gonzalez's *Digital Image Processing*[1], the algorithm is shown as follows: (suppose there are L grayscale)

1. Compute the normalized histogram of the input image. Denote the components of the histogram by p_i, I = 0, 1, 2 …, L – 1.
2. Compute the cumulative sums, $P_1(k)$, for k = 0, 1, 2,…, L – 1.
3. Compute the cumulative means, m(k), for k = 0, 1, 2,…, L – 1.
4. Compute the global intensity mean, $m_G = \sum_{i=0}^{L-1} ip_i$

5. Compute the between-class variance, $\sigma_B^2(k)$, for k = 0, 1, 2,…, L-1,

$$\sigma_B^2(k) = \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]}$$

6. Obtain the Otsu threshold, k^*, as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the various maxima detected.

7. Obtain the separability measure, η^*, by evaluating $\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2}$ at $k = k^*$

Closing

Closing includes two morphological operations in image processing, which are called dilation and erosion. We denote the dilation operation as $\oplus$, erosion as $\odot$, and closing as $\bullet$. All of these operations are applied to a binary image. It's done by operating on the original image A with another image which is called the structural element (SE) denoted here as B, such that we have:

$$A \bullet B = (A \oplus B) \odot B$$

A SE is typically represented by a "small" set (which can be different kinds of shapes) such as that shown in Figure 3 below.

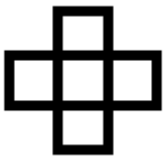


Figure 3 a typical SE.

In Figure 3 every square represents a pixel. The center of SE is called origin. What the dilation operation does is employ the SE shape to create a dilated image. Here the SE's origin is placed at every pixel location of the binary image A that has a value of '1'. At every such location, the other pixels in the *neighborhood* defined by the SE are also set to '1', no matter whether they are '0' or '1'. In contrast the erosion operation places the origin of the SE at the every pixel of A such that all non-origin values of the SE are '1 and then changes all the neighborhood values except the origin to '0'.

Combining these two operations produces the well-known closing operation in image processing. The effect of closing is likened to rolling a ball on the outside of an object to morph it [1]. This operation is useful in filling in small pixilated artefacts after thresholding to identify objects.

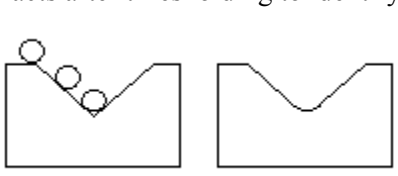


Figure 4, The effect of closing operation

Labeling

Labeling is an operation to label distinct self-contained objects of a binary image. We can define two types of connectivity, 8-connectivity and 4- connectivity.

0	1	0
1	1	1
0	1	0

1	1	1
1	1	1
1	1	1

Figure 5, 6-connectivity and 8-connectivity

The connectivity tells which pixels to included in the object.

In Matlab the labeling algorithm which is used is outlined in, Robert M. Haralick's *Computer and Robot Vision*[2], as follow:

1. Run-length encode the input image.
2. Scan the runs, assigning preliminary labels and recording label equivalences in a local equivalence table.
3. Resolve the equivalence classes.
4. Relabel the runs based on the resolved equivalence classes.

Design and Implementation

In the implementation, the SE for the closing operation must be set. Here we set it to a 'disk' of length 6

```
>> strel('disk',6)

ans =

Flat STREL object containing 109 neighbors.
Decomposition: 6 STREL objects containing a total of 22 neighbors

Neighborhood:
     0     0     1     1     1     1     1     1     1     0     0
     0     1     1     1     1     1     1     1     1     1     0
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     0
     0     0     1     1     1     1     1     1     1     0     0
```

Figure 6. SE of 'disk',6

Our tests demonstrate that the SE is significant enough to fill the gaps inside the coins.

Buildling the Simulink Model

1. Add a source file for the coin counting from Video and Image Processing Blockset →Sources → Image From File. Set the filename to "coin.jpg", the sample time to 'inf', the image signal to 'Separate color signals', the output port labels to 'R|G|B' and the output data type to 'uint16'.
2. Next convert the input color image from the file 'coin.jpg' to grayscale using Video and Image Processing Blockset → Conversions → Color Space Conversion. Set the conversion to 'R|G|B' to intensity'. Set the image signal to 'Separate color signals'.

3. Next add the Autothreshold block such that it automatically converts an intensity image from the output of the Color Space Conversion block to a binary image. Initially, set the thresholding operator to '>'. Then, uncheck 'Output threshold', 'Output effectiveness metric', 'Specify data range' and 'Scale threshold'. Finally, set the rounding mode to 'Floor' and the overflow mode to 'wrap'.
4. Now add a Close block to perform morphological closing on the binary image. Set the neighborhood or structuring element source to 'Specify via dialog' and the neighborhood or structuring element to 'strel('disk', 6)'.
5. Then add a Label block from Video and Image Processing Blockset →Morphological Operations → Label. This block will enable us to count the separate areas in the given binary images. Set the connectivity to 8. Set the output to 'Label matrix and number of labels'. In the end, set the output data type to 'uint8'.
6. Another function block we need is Insert Text block. It will help us to draw formatted text on an image or video stream. Type the text as 'Coins count: %d'. Set location source, color source and opacity source all to 'Specify via dialog'. Set the location to '[350 10]', the color value to 255, and the opacity to '1.0'. The image signal option should be 'One multidimensional signal', and uncheck 'Input image is transposed' at last.
7. Add four Video Viewer blocks from the Video and Image Processing Blockset and link them after Color Space Conversion block, Autothreshold block, Close block and Insert Text block, separately, as shown in Figure.

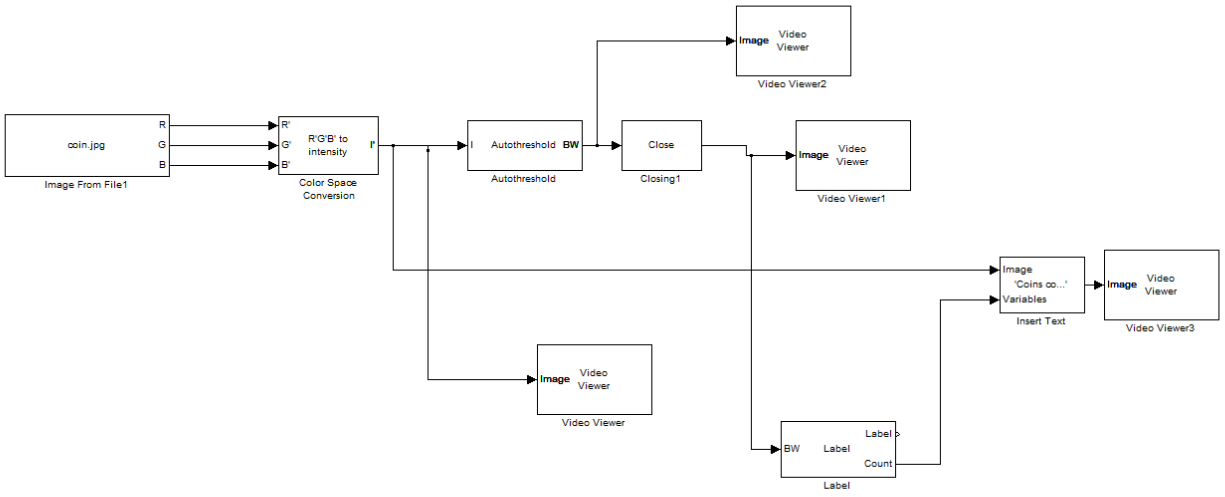


Figure 7. The Simulink model

Digital Video Hardware Implementation

1. Go to the Simulation→Configuration Parameters, change the solver options type back to "Fixedstep", solver to "discrete (no continuous states)."
2. Start with the Simulink Model for digital video edge detection and delete all the input and output blocks.
3. Under Target Support Package→ Supported Processors→ TI C6000→Board Support→DM6437EVM, found Video Capture block. Set sample time to -1.

4. Add a Video Display block. Set Video Window to “Video 0”, Video Window Position to [180, 0, 360, 480], Horizontal Zoom to 2x, Vertical Zoom to 2x.
5. Add a Deinterleave block and an Interleave block from DM6437 EVM Board Support. Link the Video Capture block to the Deinterleave block. And Link the Interleave block with a Video Display block.
6. Add a DM6437EVM Block under Target Support Package TI C6000→ Target Preferences.

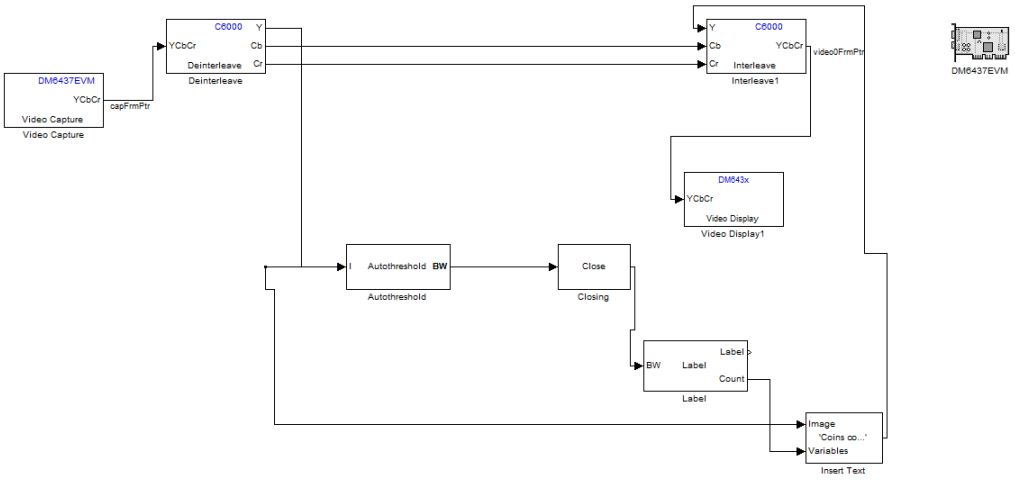
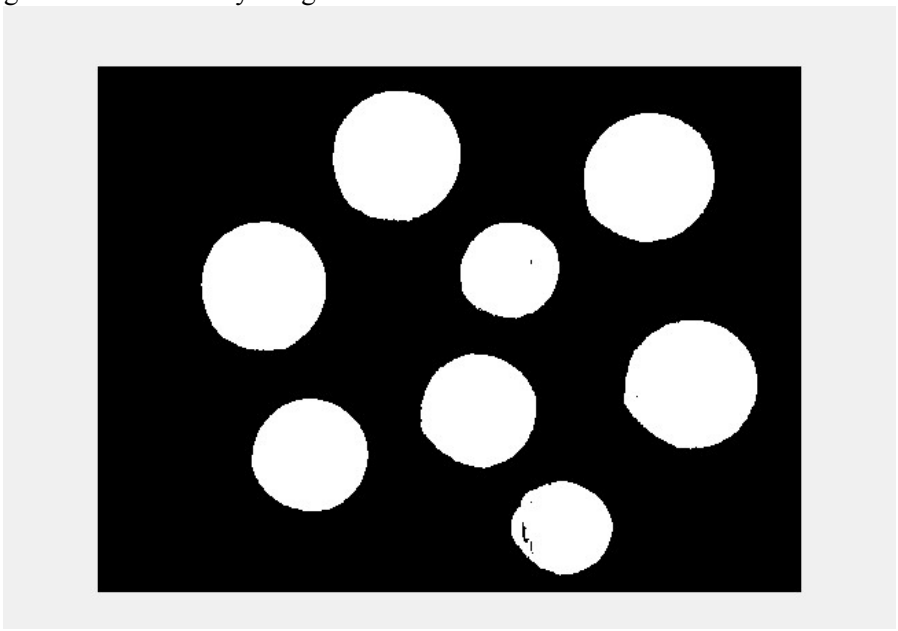


Figure 8. The hardware implementation

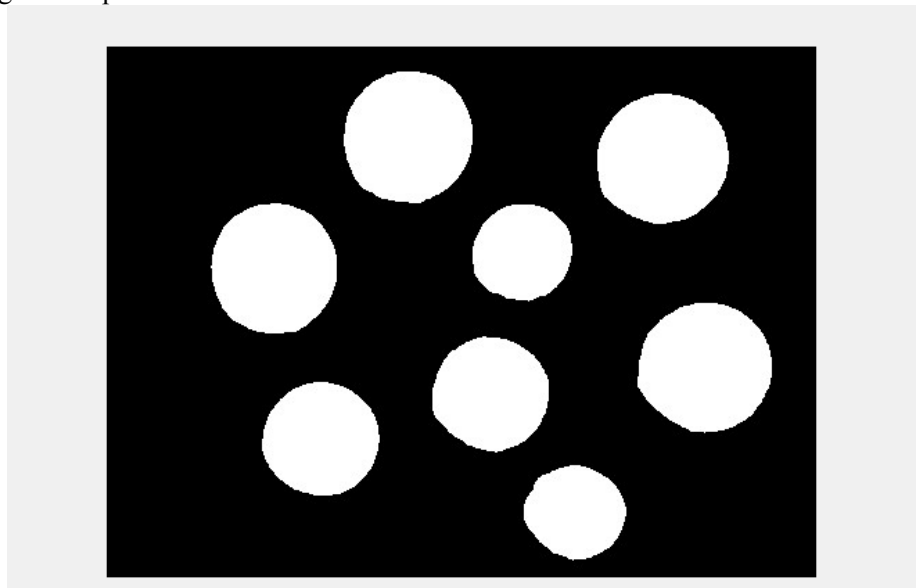
Results

1. **The captured video is converted from RGB space to intensity and the effect is as shown below:**

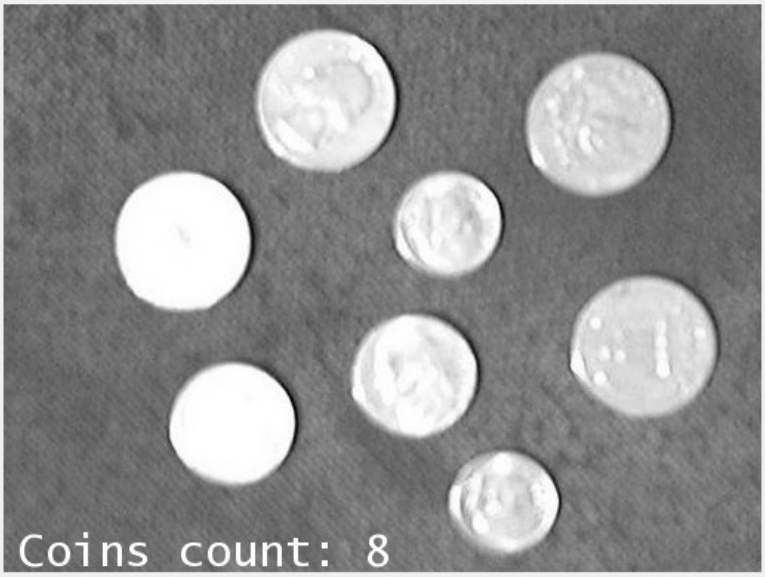
2. **This is the binary image converted from the gray-scaled image above.** As you can see, some of these testing coins in this binary image are not close areas.



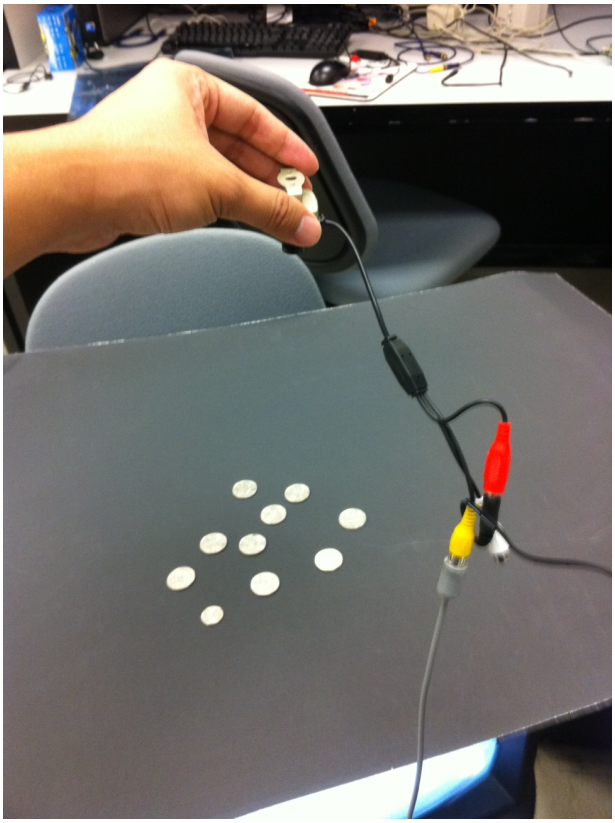
3. **We then get the image after Close algorithm.** Those little flaws in the former image are eliminated after doing close operation.

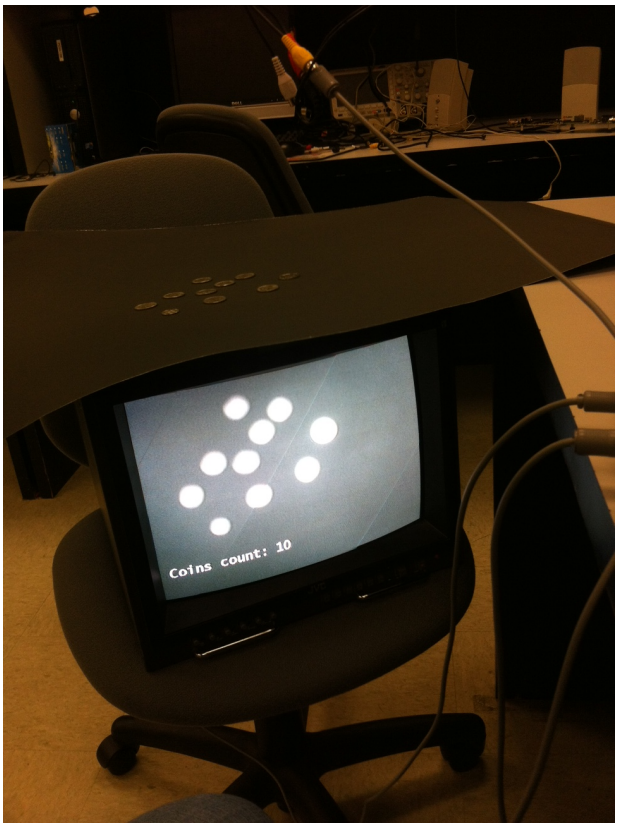


4. **The number of coins are counted and displayed in the bottom of the screen.**



5. **The real-time testing in lab:**





References

[1] Gonzalez, Rafael C., and Richard E. Woods. Digital Image Processing. 3rd ed. Prentice Hall, 2010: 769, 665-666

[2] Haralick, Robert M., and Linda G. Shapiro, Computer and Robot Vision, Volume I, Addison-Wesley, 1992: 28-48.