Hands-on Lab

Video Processing

Unlike static images, video monitors a scene dynamically by sensing changes between frames. This lab introduces video processing and leverages Scilab's Image Processing and Computer Vision (IPCV) and Scilab Computer Vision (SCV) modules. First, a simple thresholding example is provided. Next, object tracking is demonstrated. These concepts use an off-the-shelf USB camera module. Visual servoing uses frame data to command robot motions. As such, these concepts are important towards visual servoing development.

Preliminary: Scilab installation and modules

Before doing this lab, installation of Scilab and the IPCV and SCV modules must be installed. Also, the USB camera module should be connected to one's computer and tested. Some free testing software includes AMCap or eCAMView.

Concept 1: Grey-scale and Thresholded Video `scilabHelloVideo1_0a.sce`

Scilab captures 24-bit RGB video where each pixel is represented by 3 bytes (red, green and blue channels). Scilab's IPCV and SCV modules feature basic popular functions. One example is to generate greyscale version of the RGB video. Another is thresholding greyscale video. **Figure 1A** demonstrates the video feed (left column) and processed ones (middle and right). Thus is a sort of a "Hello World" example for video processing.

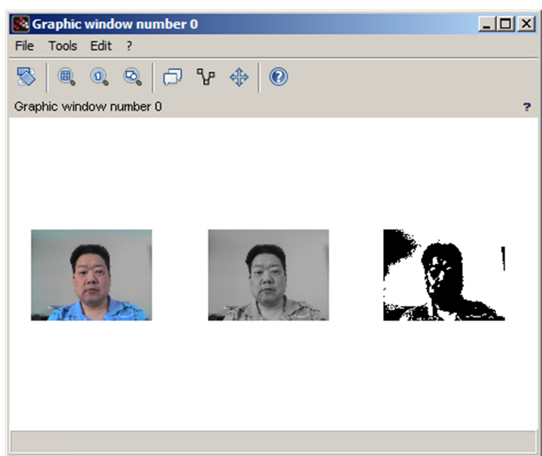


Figure 1A: Executing Scilab program `scilabHelloVideo1_0a.sce` displays live video (left), greyscale processing (middle) and thresholding (right).

Step 1: Execute Scilab and launch Editor (called SciNotes)

Assuming one has already installed ATOMS modules IPCV and SCV, when Scilab is executed, the IDE is displayed (**Figure 1B**). Click on SciNotes (red arrow) to open a new and blank canvas to start typing Scilab code (called SCE files).

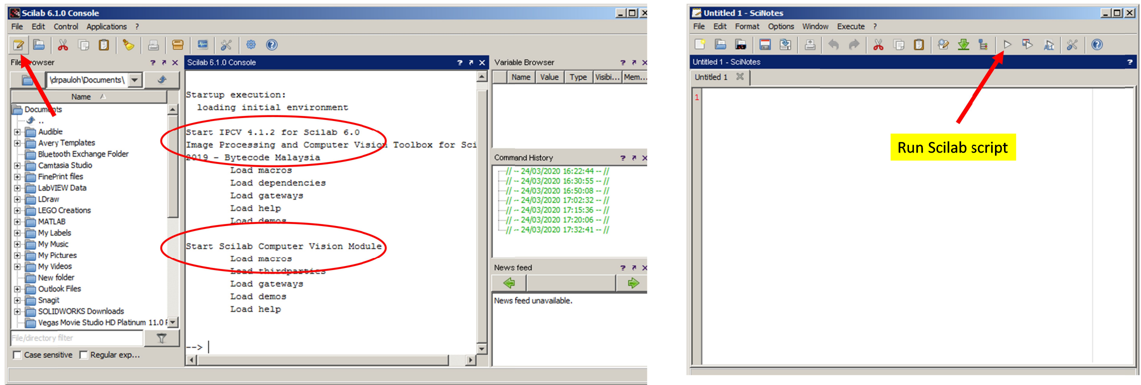


Figure 1B: Scilab IDE shows loaded ATOMS modules marked in the red ovals (left). Clicking on the SciNotes icon (red arrow in left image) will launch a blank canvass (right).

Step 2: Type scripting code into SciNotes and save as **scilabHelloVideo1_0a.sce**

```
// FILE: scilabHelloVideo1_0a.sce - Works
// DATE: 02/19/20 18:48
// AUTH: P.Oh
// REFS: Must have ATOMS modules: Image Processing and Computer Vision (IPCV)
//       and Scilab Computer Vision
// VERS: 1.0a: Basic display
// REFS: scilabHelloVision1_1b.sci
// DESC: Display what USB webcam sees: raw (color), greyscale and threshold

// (1) initialize the Scilab Computer Vision Module
scicv_Init();

// (2) Get ID of the webcam (assumes only 1 webcam connected)
//     Usually 0: computer's build-in webcam; 1: USB webcam
videoCapture = new_VideoCapture(0);

// (3) Set up a current graphic figure (window) - which will display our video

f = scf(0);

// (4) Endless loop that grabs frame, displays it, and repeats
while is_handle_valid(f)
    [ret, frame] = VideoCapture_read(videoCapture); // grab and return a frame
    if is_handle_valid(f) then
        // ret is TRUE, so display frame
        subplot(1,3,1); // display raw RGB video in column 1 subplot
        matplot(frame);

        greyFrame = cvtColor(frame, CV_BGR2GRAY);
        subplot(1,3,2); // Display greyscale version in column 2 subplot
        matplot(greyFrame);

        thresholdValue = 150; // 0 (whiter stuff becomes white)
        [thresh, thresholdedFrame]  = threshold(greyFrame, thresholdValue, 255, THRESH_BINARY);
        subplot(1,3,3); // Display thresholded video in column 3 subplot
        matplot(thresholdedFrame);
    end // end if
end // end while

delete("all"); // kill all frames
```

Figure 1C: SciNote file **scilabHelloVideo1_0a.sce**

The SCE file comments four steps for implementing and displaying video. One observes that Scilab code looks similar to C programming as well as Matlab scripts. First, all video processing begins with initiating the SCV module by calling the function `scicv_Init()`. Second, the camera is specified by calling `new_VideoCapture(0)`. As commented, many laptops have a built-in camera. Thus "0" would invoke the computer's default camera. Since this lab uses a USB camera module, one may need to change this to `new_VideoCapture(1)`. This function returns a user-defined handle, which for this example, is named `videoCapture`. Third, the video's display window is setup by calling the function `scf(0)`. This function sets the current graphic figure as the one to display in. This call returns a user-defined handle, which in this example is called `f`. The last step is an endless `while` loop. This is where one would put any video processing statements.

Step 3: Filling the `while` loop - Implement Greyscale conversion and Thresholding

The endless while loop makes several function calls. The first is to capture one frame from the video feed by calling the function `VideoCapture_read(videoCapture)`. By using the previously defined handle `videoCapture`, the frame is stored in the variable `frame`. The `subplot` and `matplot` functions in Scilab mimic those in Matlab; here the raw RGB frame is displayed in the first column of the current graphic window (left image in **Figure 1A**).

The SCV function `cvtColor` is used to convert images. There are several options and `CV_BGR2GRAY` is the SCV-defined variable for converting the RGB frame to greyscale. The function returns a handle that is stored in the user-defined variable named `greyFrame`. Again, subplot and matplot are used to display this greyscale frame in the second column of the current graphic window (center image of **Figure 1A**).

SCV also has a function for thresholding called `threshold`. This function takes as input, the frame one wishes to threshold (which was called `greyFrame`), compares it to a user-defined threshold value (which was called `thresholdValue` and set to `150`). The additional inputs specify that the maximum value of a pixel value (255 in this case), and that a binary image (black or white) is to be generated (using the SCV defined variable `THRESH_BINARY`). The resulting thresholded frame is stored in user-defined handle, named `thresholdedFrame` in this example. Again, subplot and matplot are used to display the thresholded image in column 3 of the current graphic window (right image of **Figure 1A**).

When the user terminates the program, the while loop exits and the graphic windows are deleted and release memory.

Step 4: Run the Scilab script

Clicking on the Execute button (see red arrow in **Figure 1B** right) will run the SCE script and should display the 3 images on a single row, as shown in **Figure 1A**.

Congratulations! You can capture, processing and display Video!

Concept 2: Object Detection with Static Images - `sciLabTracking1_0a.sce`

In lecture, the sum-of-square differences (SSD) similarity measure was introduced. The SSD is commonly used in image and video processing to track objects. As such, it is a built-in function in many vision software packages. Both Scilab and Matlab have module and toolboxes that include the SSD. This SSD function will be first demonstrated with a static image in Scilab.

PixelFormer was used to create greyscale (i.e. 256-color plate 8 bits per pixel) pixel maps. **Figure 2A** left and right respectively are the 50x1 image and 1x1 template pixel maps. The annotated text boxes and red arrows just show relevant pixel locations. These locations were confirmed by moving the mouse over these pixel boxes. Pixelformer `File – Export` was then used to save these pixel maps as PNG files

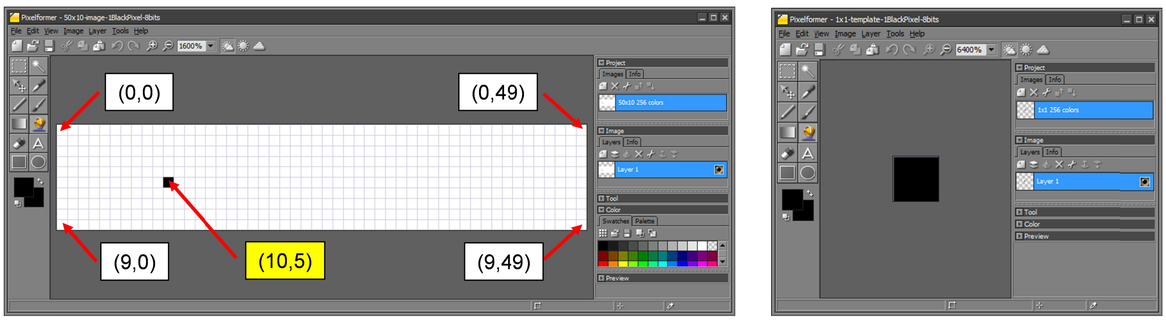


Figure 2A: 50x10 image file `image1BlackPixel.png` (left) and a 1x1 template file `template1BlackPixel.png` (right)

Step 1: Type scripting code into SciNotes and save as `sciLabTracking1_0a.sce`

```
// FILE: sciLabTracking1_0a.sce
// DATE: 03/18/20 16:02
// AUTH: P.Oh
// VERS: 1_0a: SSD tracking of 1x1 black pixel template thru a 50x10 image
// DESC: Goal: Find object in an image.
scicv_Init();
img = imread("M:\00courses\scilabVideo\image1BlackPixel.png");
img_template = imread("template1BlackPixel.png");
img_result = matchTemplate(img, img_template, CV_TM_SQDIFF); // 0 = match
disp("Result: number of Rows:");
disp(Mat_rows_get(img_result));
disp("Result: number of Columns:");
disp(Mat_cols_get(img_result));
/* uncomment if wish to all values
disp("img_result: entire");
disp(img_result(:,:));
*/
[min_value, max_value, min_value_loc, max_value_loc] = minMaxLoc(img_result)
disp("min_value =");
disp(min_value);
disp("location in image:")
disp(min_value_loc);
delete_Mat(img);
delete_Mat(img_template);
delete_Mat(img_result);
```

Figure 2B: `sciLabTracking1_0a.sce` implements SSD tracking

Like in Concept 1, `scicv_Init` is first called to launch Scilab's image processing ATOMS module (called SCV). The `imread` function is used to read the desired image files. If needed (e.g. files paths were not setup), one should explicitly show the drive and folder location of the image file (yellow highlight).

The SCV function `matchTemplate` takes the image, template and desired similarity measure as inputs. SCV has a defined constant named `CV_TM_SQDIFF` (yellow highlight) which implements the SSD equation (shown in lecture) for `matchTemplate`. The results of matchTemplate are stored in a Scilab `MAT`-type variable. This program names this variable `img_result`.

Beyond the scope of this concept, `MAT` variables are not simple 2-dimensional arrays. Rather, they are defined in `OpenCV` and contain much more information like header information and pointers to the memory locations of the image pixels. The important point to note is that Scilab's SCV calls `OpenCV` libraries. The beauty of this is that one does not have to go thru the burden of installing `OpenCV` separately.

To show the pixel location in the original image with the best match, the Scilab function `minMaxLoc` is used. Recall that with SSD, the value of 0 means a perfect match.

Step 2: Execute the program

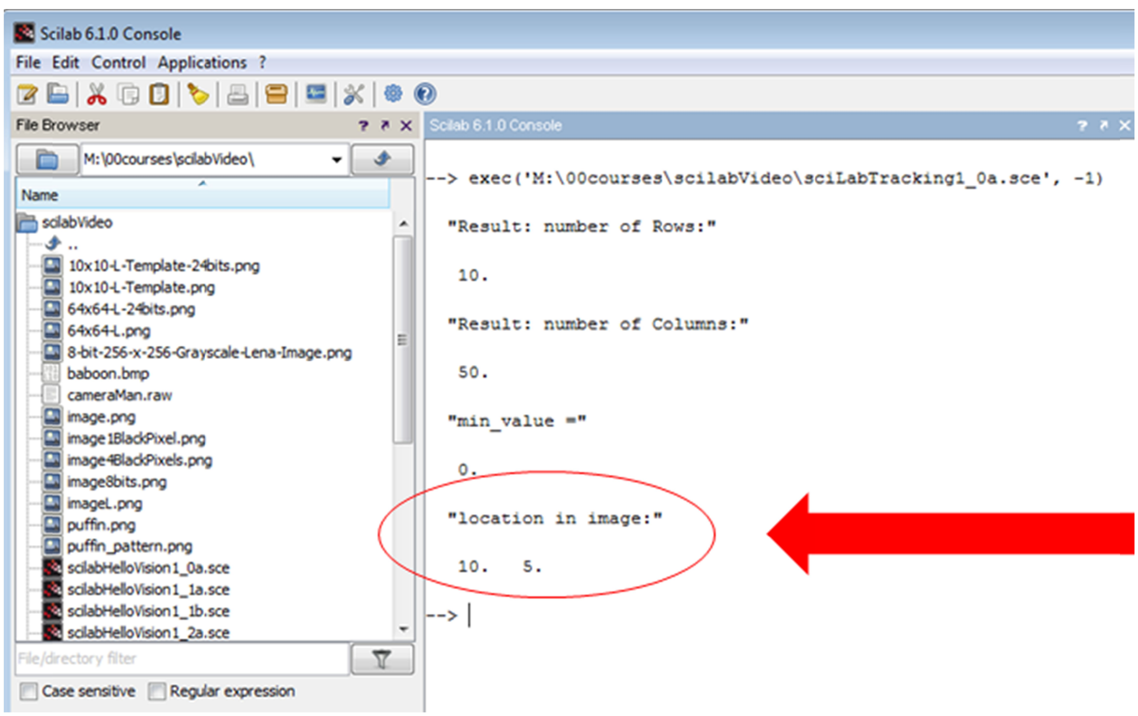
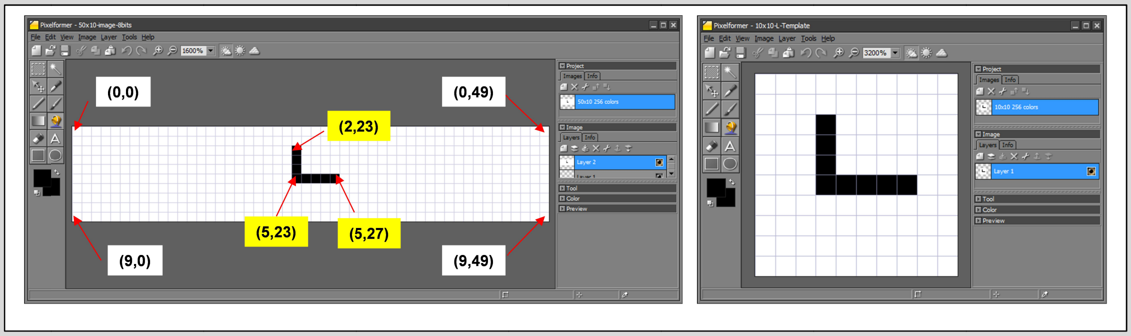


Figure 2C: Result of executing **`sciLabTracking1_0a.sce`** shows output of `matchTemplate`

Referring to **Figure 2A**, we know the 1x1 black pixel should in the image at row 10, column 5 i.e. (10x5). Indeed, Figure 2C shows the SSD shows an exact match (minimum value of 0) at that location.

Congratulations! You can track objects using Scilab's matchTemplate which actually calls OpenCV libraries

Exercises



In lecture, the above (50x10) image and (10x10) template were introduced. The notes showed the sliding process to comprehend why matchTemplate yielded a (20, 0) location result.

1. Use Pixelformer to create your own 50x10 and 10x10 pixel map and corresponding PNG files. For example, replace the L-shaped figure above with say, an X-shaped one. Annotate your pixel map (e.g. cut-and-paste the figure in PPT) with relevant pixel locations. Run your SSD program to calculate the match result. Compare with sliding figures that the result indeed is the location of the template in the image file

2. Create a 50x20 pixel map and repeat the “1” above