**Hands-on Lab**

**Video Processing**

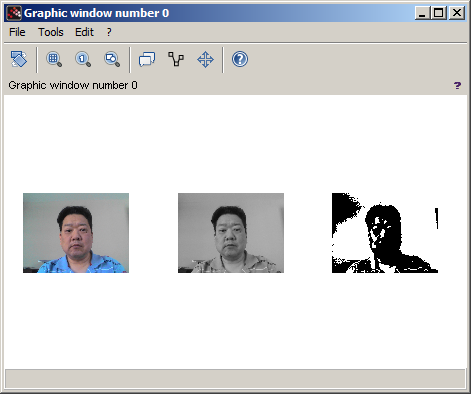
Unlike static images, video monitors a scene dynamically by sensing changes between frames. This lab introduces video processing and leverages Scilab’s Image Processing and Computer Vision (IPCV) and Scilab Computer Vision (SCV) modules. First, a simple thresholding example is provided. Next, object 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](https://drive.google.com/drive/folders/10qdEM0TAoTqhkjfphi-SdeKJkJqOG9xX) or [eCAMView](https://www.e-consystems.com/e-CAMView-Windows-camera-streaming-app.asp).

**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**. ecute button (see red arrow in aphic windows are lloted ure 1A).mage in in this example., and rst column (left image in Figure

Congratulations! You can capture, processing and display Video!

**Concept 2:** Object Detection - **areaCentroid1\_0.c**

Blah: Need to give white 32 x 32 base plate and associated Lego pieces

I need to show how this ROI object detection works (perhaps thru a PPT lecture).

I need to demo the detection of a black Lego piece on the 32 x 32 baseplate. I might have to talk about pinhole camera model and mapping from image space to baseplate space.

**Concept 3:** Drawing white box **- whiteBox1\_0.c**

Exercises

* 1. Create an output image with a white rectangle centered in the image (like shown in **Figure 3D right**), defined by your desired rectangle height and width.
  2. Create an output image with a white box (rectangle height and width are the same), centered in the image.
  3. Modify your program in 3.2 to also have diagonal lines spanning from the top-left corner to the bottom-right corner, and from the top-right corner to the bottom-left corner.

Congratulations! You can read input images, perform calculations, and draw output images – the basics of Image Processing and Computer Vision!