**Hands-on Lab**

**Video Processing**

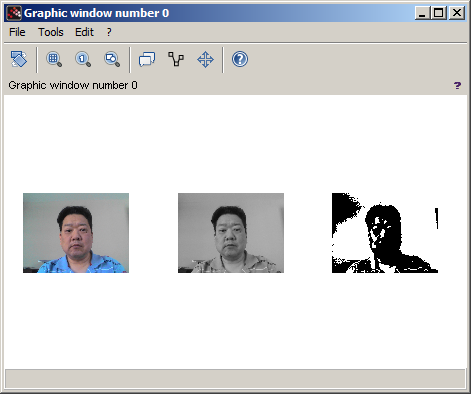
Unlike static images, video monitors a scene dynamically by sensing changes between frames. This lab introduces video processing and leverages Scilab’s Image Processing and Computer Vision (IPCV) and Scilab Computer Vision (SCV) modules. First, a simple thresholding example is provided. Next, object 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) 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:** Implement Greyscale conversion and Thresholding

Exercises

* 1. Write a program to threshold a RAW grayscale image (e.g. cameraMan.raw) so that only the whitest pixels remain white.

* 1. Write a program to reads a RAW mage file (e.g. cameraMan.raw) and outputs the inverse (i.e. a negative).

**Concept 2:** Areas and Centroids - **areaCentroid1\_0.c**

**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!