EW309 Computer Vision

Acquiring Images

Goal

The syntax for acquiring images from a digital camera is dependent on the camera, drivers, and software selected for a given project. For this class, we need to interface a USB webcam with MATLAB to quickly acquire digital images.

Required Software

The remainder of this document and much of this course assumes:

1. Use of MATLAB 2017b or later
2. The following MATLAB support package(s) are successfully installed:
    a. Image Processing Toolbox
    b. MATLAB Support Package for USB Webcams
3. The following MATLAB function(s) exist in your current MATLAB directory (or have been added to your MATLAB path):
    a. initWebcam.m (https://github.com/kutzer/WRC_MATLABCameraSupport/archive/master.zip)

Installation Instructions

To install MATLAB and most support package(s), you will need to login to your MathWorks Account. The email address for this account will be your USNA email address. To install, use the following steps (skip to Step 3 if MATLAB is already installed).

1. Login to MathWorks (https://www.mathworks.com/login)
2. Download and install MATLAB
3. Open MATLAB as an Administrator
4. Run "supportPackageInstaller"
5. Install necessary support packages

Initializing the Camera

Camera selection (when applicable) and initialization is wrapped into a single MATLAB function:

```
[cam,prv] = initWebcam;
```

Calling initWebcam.m with this syntax will do the following:

1. Initialize a webcam object with a handle contained in the variable "cam"
2. Initialize an image object with a handle contained in the variable "prv"

Acquiring Images

Given a valid image object handle for your webcam preview (e.g. “prv”), images can be quickly and easily acquired using the “get/set” syntax:

```
im = get(prv,'CData');
```

Or using the “dot” syntax:

```
im = prv.CData;
```

Note that “CData” is an abbreviation for “Color Data.”

Observations

In MATLAB, color images are represented as three-dimensional arrays. When using the RGB (red, green, blue) color space, the dimensions of the image will be $M \times N \times 3$. In this context, M represents the number of pixels vertically (i.e. the number of rows), and N represents the number of pixels horizontally (i.e. the number of columns). The three layers or channels represent the pixel intensities in the red, green, and blue spectrum. For example, given an image named “im”, the pixel intensity in the red spectrum can be isolated using:

```
im_Red = im(:,:,1);
```

the pixel intensity in the green spectrum can be isolated using:

```
im_Green = im(:,:,2);
```
and

the pixel intensity in the blue spectrum can be isolated using:

```
im_Blue = im(:,:,3);
```

Plotting

Generally, images can be plotted in MATLAB using imshow.m. For the purposes of code development in this course, it is always advisable to capture and use the handles of the graphics objects you create. This will both speed up your code and eliminate inadvertent misplacement of graphics objects.

In MATLAB, all graphics have an underlying dependency whether you recognize it or not. A Basic example of MATLAB graphics dependency is given in Fig. 1.

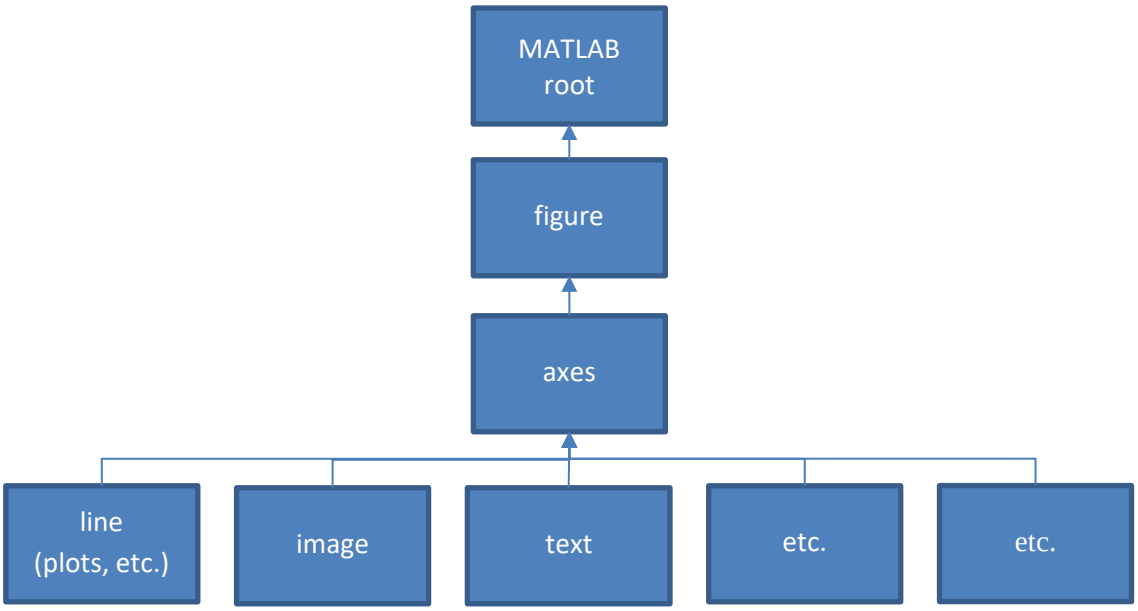

Figure 1: A Basic example of MATLAB graphics dependency. Arrows depict dependency and are shown pointing from "child" to "parent."

A figure object (i.e. the window containing your plot) whose handle is contained in the variable "figEXAMPLE" can be created using the following:

```
figEXAMPLE = figure('Parent',0);
```

where 0 is universally the handle of the MATLAB root. An axes object that is contained in (i.e. a dependent or child of) this figure can be create using:

```
axsEXAMPLE = axes('Parent',figEXAMPLE);
```

Here, the handle of the axes object is contained in the variable "axsEXAMPLE." This syntax is generally applicable when adding items to the axes, additional axes or panels to the figure, etc. As an example, an image contained in the variable "im" can be added to the axes using:

```
imgEXAMPLE = imshow(im,'Parent',axsEXAMPLE);
```

where imgEXAMPLE is the handle of the image object.

MATLAB Graphics Object Properties

Each graphics object in MATLAB can be updated by changing the properties associated with a given handle. The most common property that we change when plotting information is the "NextPlot" property associated with axes objects. We commonly do so using the command:

```
hold(axsEXAMPLE,'on');
```

or, when ignoring graphics objects (which is rarely advisable but unfortunately all too convenient):

```
hold on
```

This command changes the "NextPlot" property from the default value of "replace" to the value "add." The result is an axes object that will add graphics objects rather than replacing any/all existing graphics objects that are added as children. This property can also be changed using acquired using the "get/set" syntax:

```
set(axsEXAMPLE,'NextPlot','add');
```

Or using the "dot" syntax:

```
axsEXAMPLE.NextPlot = 'add';
```

Some object properties, like "NextPlot", have a limited set of discrete values that are valid; while others have numeric or array values. To get all properties and current property values associated with a given object (e.g. "axsEXAMPLE") you can use:

```
get(axsEXAMPLE);
```

Similarly, to get the discrete options for all properties, you can use:

```
set(axsEXAMPLE);
```

Bringing It All Together

Take the time to combine everything you have learned into a single MATLAB script to accomplish the following:

1. Initialize your camera and acquire an image
2. Create a figure named (use the "Name" property) "EW309 – Camera Overlay Test"
3. Create an axes in your figure and adjust the "NextPlot" property
4. Add your image to the axes
5. Add a crosshair to the axes using two independent line objects (created using the line or plot functions)
6. Create an indefinite while loop (using `while true`)
    a. Get an updated image
    b. Update your image object color data (use the "CData" property) with the new image
    c. Allow your plot to update using the `drawnow` command