A Gentle Introduction to Computer Vision using SimpleCV

Katherine Scott

SightMachine

kat@sightmachine.com

March 11, 2013

Outline

Get Started!

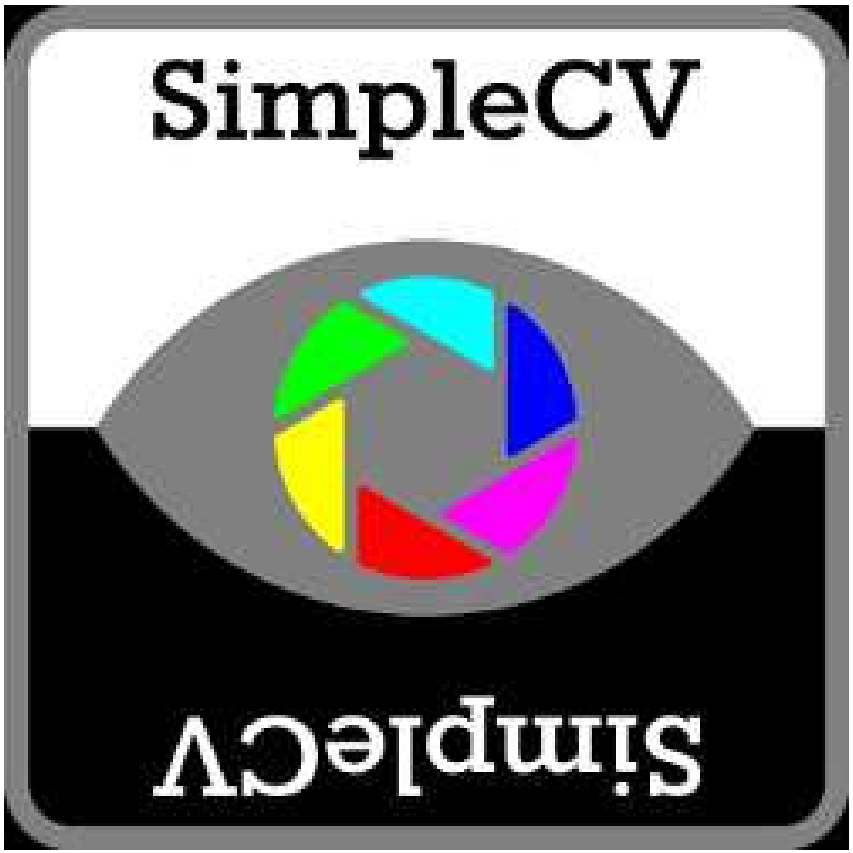
There are a lot of dependencies for SimpleCV and it is a bit tough for beginners. We've brought disks that are ready to go!

- Windows / Linux
  - Boot from USB drive.
  - Alternatively install VirtualBox and the image.
  - `https://www.virtualbox.org/`
- Macs
  - Newer macs are persnikety about booting from a USB drive.
  - Install virtual box and the ISO and go to town.
- When you get home install from SuperPack or preferably source libs.
  - take awhile and is not a perfect science.
  - `https://github.com/ingenuitas/SimpleCV`
  - If you want to contribute this is a great place to start.

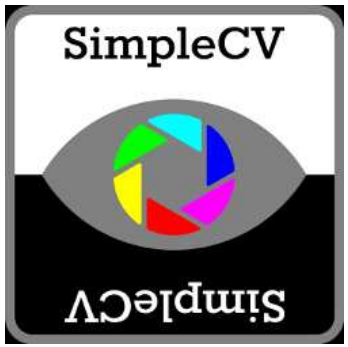
About the tutorial

- It will be a lot of live coding. I'll lead, you follow along.
- If you have a question feel free to interrupt.
- If you are having an issue raise a hand. Asistants will help you.

What Makes Up SimpleCV?



SimpleCV != OpenCV



- OpenCV is really busy, we help by wrapping python.
- We add lots of other fun stuff (OCR, Barcodes, etc.)
- We are not competing, we are complementing.
- Purposes are different. Python is great for prototyping. C++ great for embedded.

Core Dependencies

- OpenCV Python Bindings
- Numpy
- SciPy
- SciKits Learn and Orange
- PyGame (this is going away)
- Python Imaging Library (PIL)
- ipython
- PIL (Python Imaging Library)

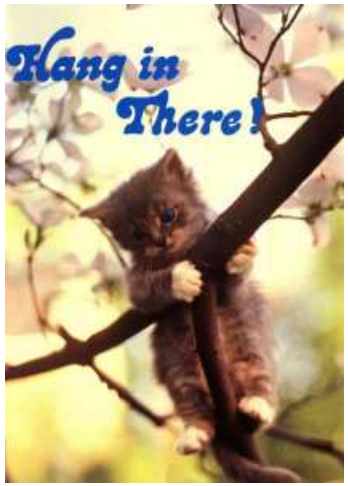
Optional Dependencies

- Barcodes- Zebra Crossing ZXIng
- Optical Character Recognition (OCR) - Tesseract
- Beautiful Soup
- Kinect Support - freenect
- Unit Tests - nose
- Web Stuff - flask / CherryPy
- Arduino - pyfirmata
- Many Many Many more.

This is why we put everything in a superpack / virtual box / bootable drive

- Just get to the core library functions.
- We encourage you to install the full library when you get home.
- Help is available if you need it.

Getting Help after the tutorial.



- Primary Source: `http://help.simplecv.org/questions/`
- Documentation `http://www.simplecv.org/docs/`
- Tweet at us: @Simple_CV
- Another Good Resource: `http://www.reddit.com/r/ComputerVision`

On the Printed Page

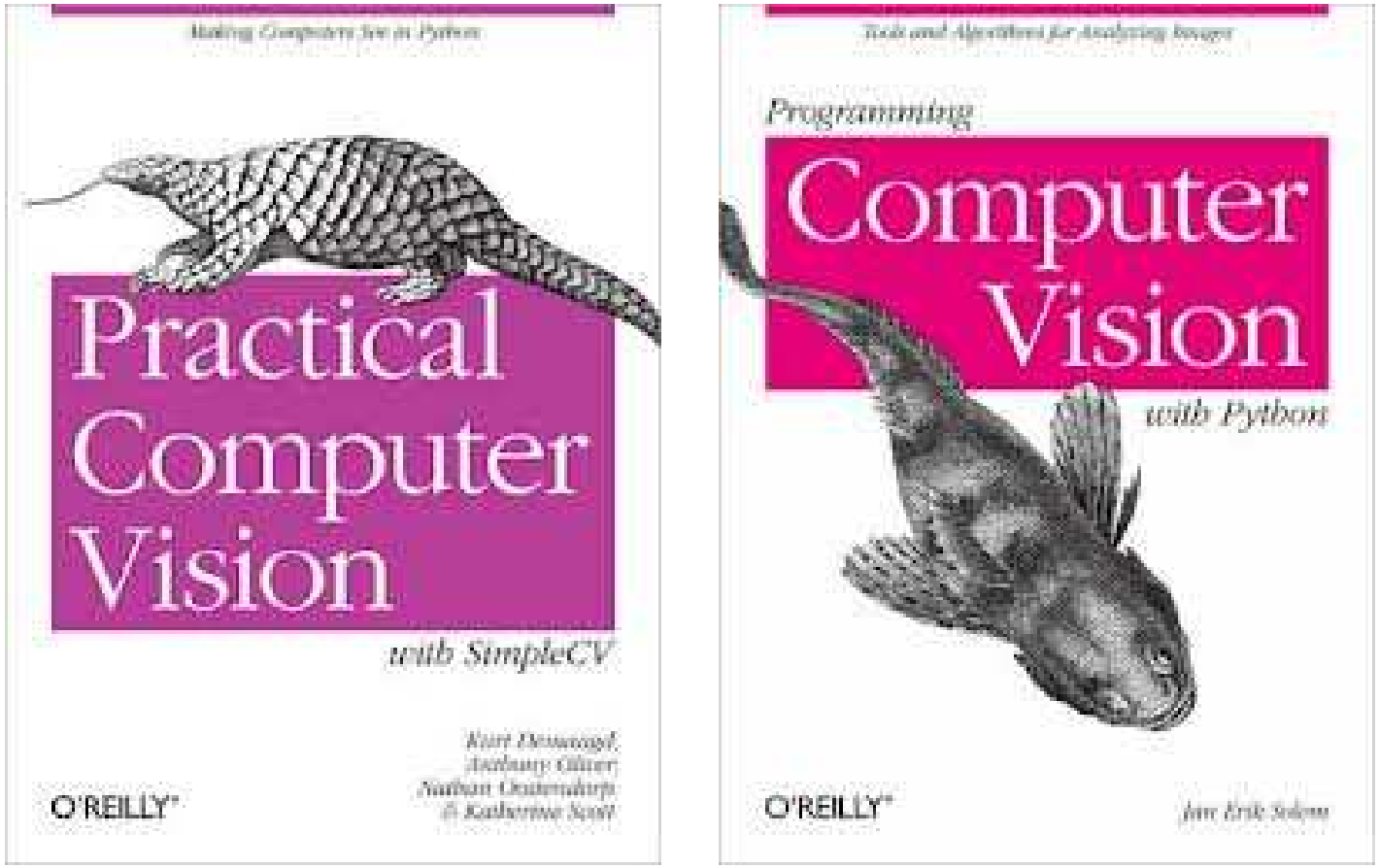


Figure: Two books about using Python for Computer Vision

So why are we doing this?

- We are really nice people who believe in Python and Open Source.
- We are trying to disrupt industrial quality control systems.

Early Prototypes

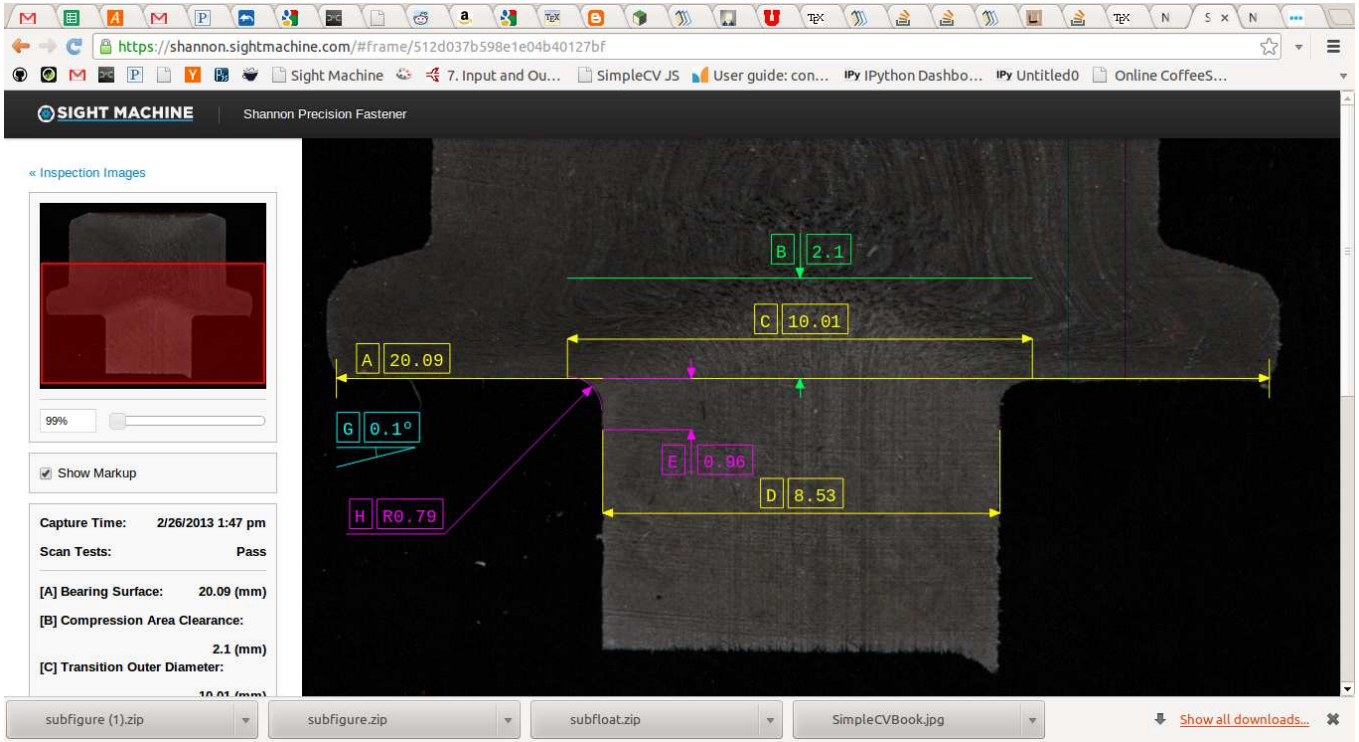


Figure: Early Customer - Industrial Fastener Morphology and Metallurgy

Early Prototypes

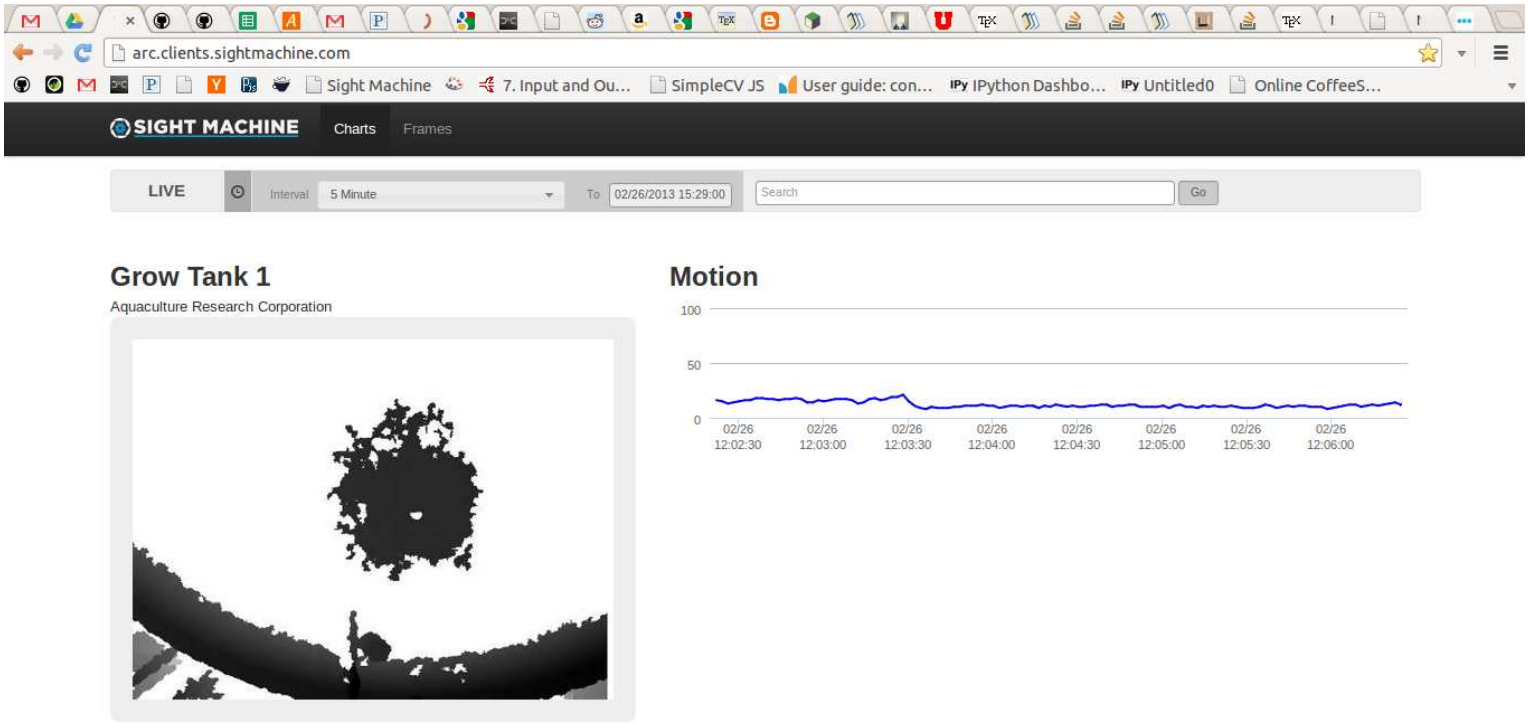
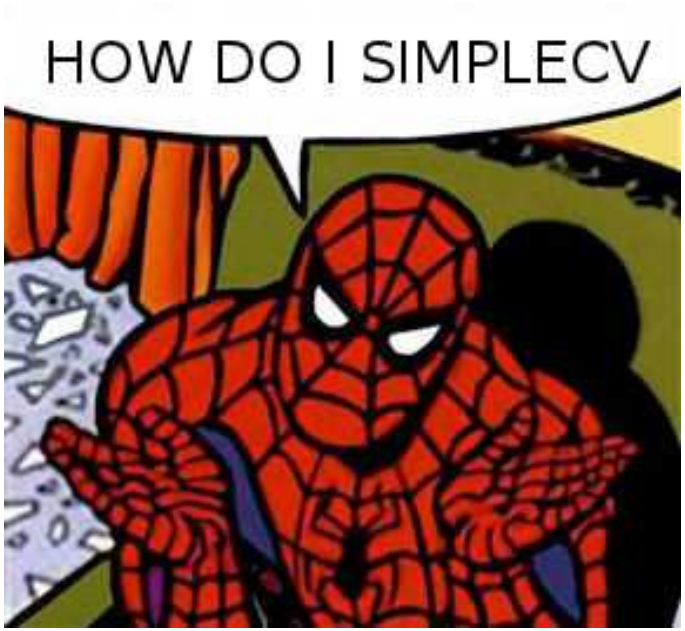


Figure: Early Customer - Aquaponics Research Facility

How do I SimpleCV?



Where do I write my code?

So how do I SimpleCV?

- In a python file, just like any other library.
- In a command line REPL like iPython.
- In the browser using iPython Notebooks (we'll use this today).

We really like iPython. It is kinda like using Matlab without the $ 5000 per seat license cost.

How does fit into a work flow?

At SightMachine we roughly use these three tools for different parts of our workflow.

Tool	**Uses**
iPython REPL	Prototypes, Sanity Checks, Etc
iPython Web Notebook	Testing and Development
Python Files	Deployment Code

Table: SimpleCV Workflow

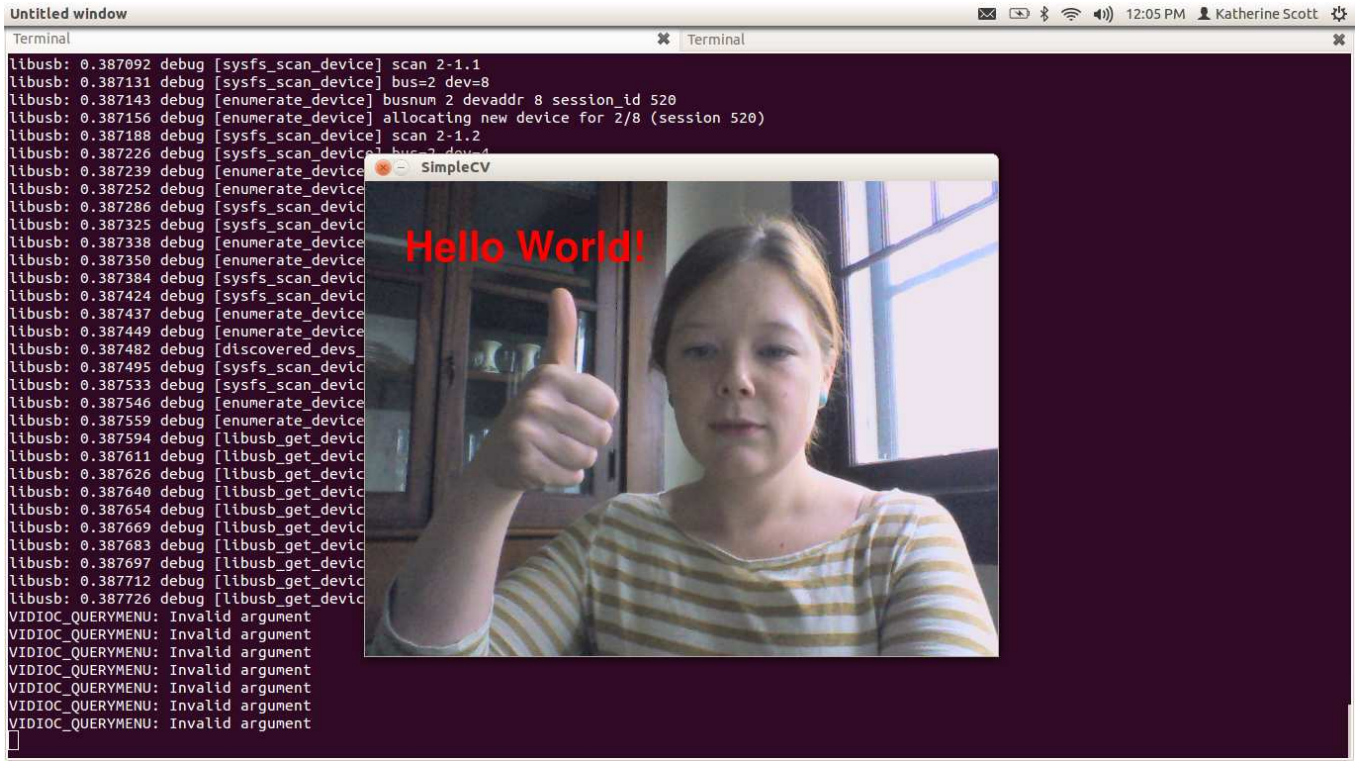
SimpleCV Hello World as a Script

Example (HelloWorld.py)

```
1  from SimpleCV import Image, Display, Color, Camera
2  cam = Camera(0) #Get the first camera
3  disp = Display((640,480)) # Create a 640x480 Display
4  while( disp.isNotDone() ):
5      img = cam.getImage() # get an image
6      # write text at 40,40 font_size 60pts, color is red
7      img.drawText("Hello World!",40,40,
8                   fontsize=60,color=Color.RED )
9      img.save(disp) # show it
10
```

How do I run Hello World?

- Run the py file with *python HelloWorld.py* in the command.
- Close it by pressing *esc* or *ctrl* − *c*



The SimpleCV Shell - Custom iPython REPL

Sometimes you just want to test an idea without writing a full script. For this reason we created the SimpleCV shell, which is a custom ipython instance. The SimpleCV shell will allow you to:

- Test your ideas in a REPL similar to Matlab.
- Access the SimpleCV documentation.
- Import modules that you are working with to test.
- Run through an interactive tutorial.

Starting the SimpleCV Shell

In OSX and Linux just type *simplecv* at the command line. On Windows you just click on the SimpleCV icon.

Example (Shell Basics)

```
+-------------------------------------------------------------+
 SimpleCV 1.3.0 [interactive shell] - http://simplecv.org
+-------------------------------------------------------------+
Commands:
"exit()" or press "Ctrl+ D" to exit the shell
"clear" to clear the shell screen
"tutorial" to begin the SimpleCV interactive tutorial
"example" gives a list of examples you can run
"forums" will launch a web browser for the help forums
"walkthrough" will launch a web browser with a walkthrough
Usage:
dot complete works to show library
for example: Image().save("/tmp/test.jpg") will dot complete
just by touching TAB after typing Image().
Documentation:
help(Image), ? Image, Image ? , or Image() ? all do the same
"docs" will launch webbrowser showing documentation
```

SimpleCV Shell Like a Boss

- Putting a ? in front of a class or method will give you documentation. The "/" key will let you search.
- iPython has tab completion for methods.
- Up arrow will give you previous commands.
- %paste will let you paste formatted code.
- Other cool stuff can be found by googling iPython magic commands.

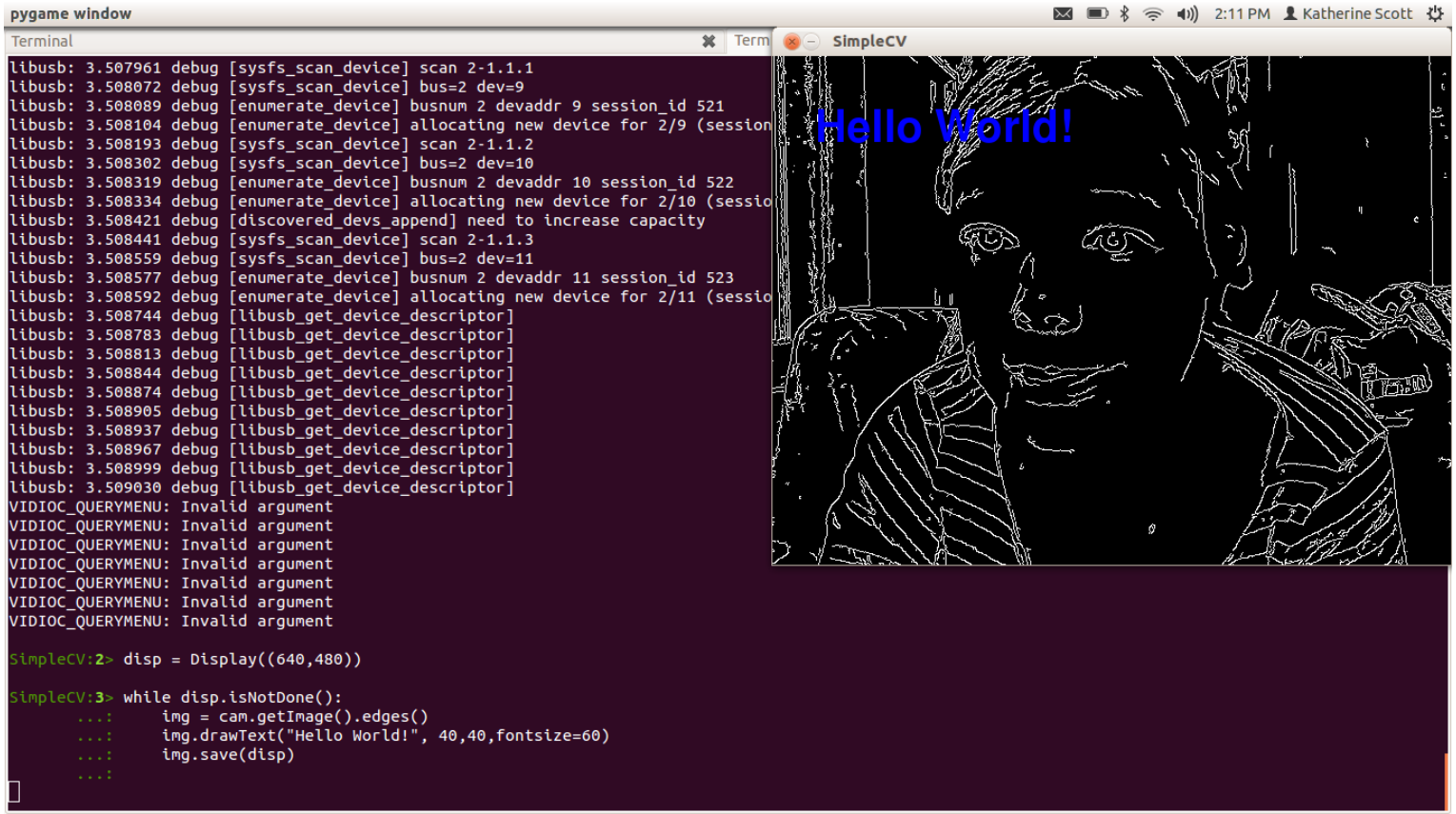
Let's repeat Hello World in SimpleCV Shell

Example (In the SimpleCV shell)

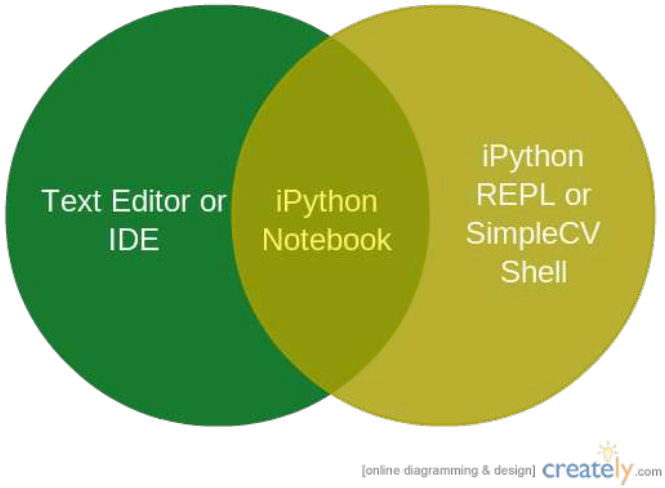
```
SimpleCV:1> cam = Camera()
SimpleCV:2> disp = Display((640,480))
SimpleCV:3> while disp.isNotDone():
       ...:     img = cam.getImage().edges()
       ...:     img.drawText("Hello World!",40,40,fontsize=60)
       ...:     img.save(disp)
       ...:
SimpleCV:4> exit
```

- Just push return after each line.
- iPython will do tabbing in the while loop.
- *esc* to quit or *ctrl* − *c*.
- type "exit" to quit.

Yes, it really is that simple.

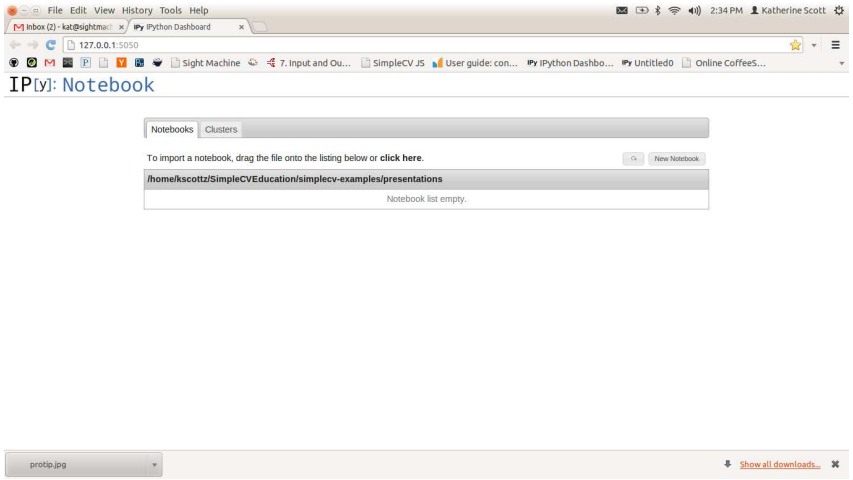


Why use iPython Web Notebooks



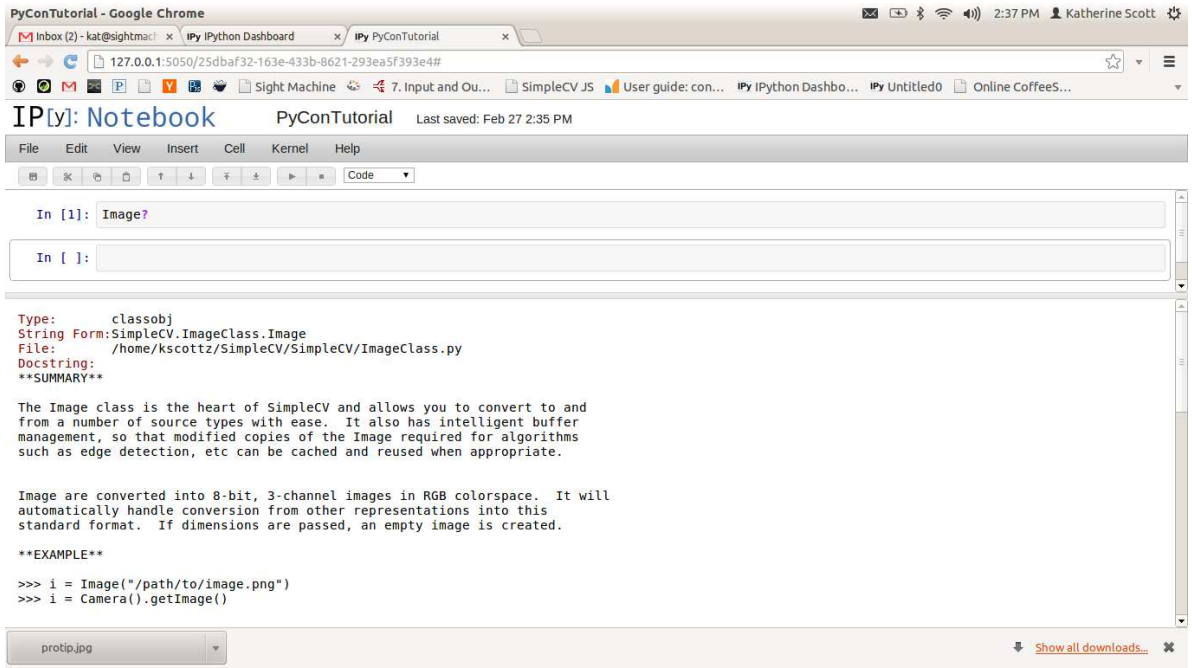
- Web notebooks give you the best features of an IDE and a REPL.
- You can edit chunks of code, and then execute them in series.
- iPython is still v. 0.1.4 but it is getting good fast.

How do I use the notebook?



- From the shell just type *simplecv notebook – – pylab inline*.
- The –pylab inline is optional but it pulls in matplotlib which is handy.
- You will get to a dashboard to create a new notebook.
- By default notebooks are in the path where you start ipython.

How do I use the notebook?



- Everything we mentioned about the SimpleCV shell still holds.
- Magic commands, inline documentation, etc. still work.
- *enter* starts a new line.
- *ctrl* – *enter* executes a line.

Caveats about iPython Web Notebooks

- iPython Web Notebooks are still version 0.1.4
- **There is no auto-save. Get in the *ctrl* – *s* save habit.**
- If you edit a module you import you must restart the core.
- Minimal editing support. No find/replace.
- The core can sometimes crash on large images.
- The notebooks hold on to data by default. This can fill up your version control system fast. Try the download as python command from the gui.

Image Loading Basics II

- You can get the image file name using *img.filename*
- Images can also come from appropriately shaped numpy arrays.
- PIL and OpenCV images can also be passed into the image.
- Can also take a URL to an image.
- The **img.getEXIFData()** command can show jpg EXIF data.

Saving an Image

- The **img.save()** command is used to save images.
- You can save as just about any format, PNG, JPG, WebP, etc.
- Calling save with no parameters saves it to temp directory.
- Using the params flag you can set compression, e.g. set compression quality.

Using a USB Camera

- Most usb cameras use the **Camera** class.
- The camera class takes a camera index, *usually* this is the order cameras are plugged into the computer.
- The camera also has properties that you can get and set, or use a propmap dictionary to set.
- **Support for camera properties is vendor specific and spotty at best.**
- Cameras also have a **threaded** parameter. Set this to false to run multiple cameras.

Using a USB Camera II

- **Camera.getImage()** will return the current image.
- **Camera.getAllProperties()** will return the cameras properties.
- **Camera.getProperty() and Camera.setProperty()** may let you set properties. *This is highly vendor dependent and usually poorly documented.*

Briefly: Other Cameras

- Kinect - Depth Camera
  - Uses freenect drivers, not the OpneNI drivers.
  - **Kinect.getImage and Kinect.getDepth**
  - Note that these aren't well calibrated together.
- JpegStreamReader - IP Cameras
  - Give it a url to camera's web feed, and scrape images.
  - Getting the URL straight can be tricky.
- Virtual Camera
  - A virtual camera that pulls from a directory full of images, or video.
  - Interface to video files for processing.

Briefly: Other Cameras

- Document Scanners
  - SANE compatible devices.
  - Allow you to set resolution and ROI.
- Digital Camera
  - Uses Piggy Photo Library
  - Works with most DSLRs and point and shoots.
- AVT Camera
  - Professional digital imaging cameras with interchangeable optics.
  - Fine grain control of camera parameters.

Image Sets

ImageSets are lists of images. They are great for aggregating datasets.

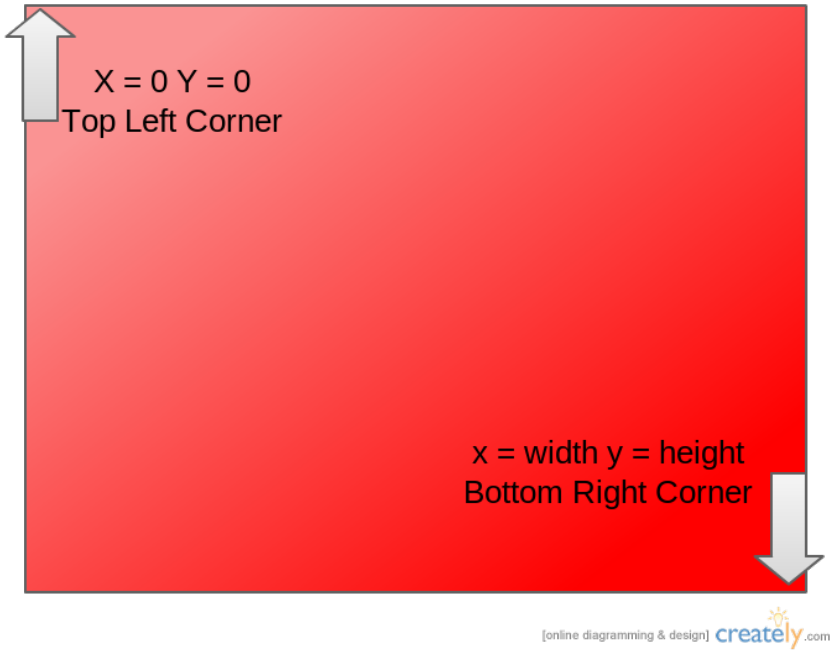
- By default load all image files in a directory.
- Can iterate over the list using list comps or for loops.
- Using the BeautifulSoup library can download sets from google.
- Can save image sets to directories, or animated gifs!
- The show command works just like on image class.
- Can apply averages to images.
- More coming soon.

ImageSet Example

Example (Image Sets)

```
mySet = ImageSet()
mySet.download('cats') # download cats
mySet.show() # show them to us
avgCat = mySet.average() # get the avg cat
avgCat.show() # show the avg cat
mySet[3].show() # show the third kitty
resized = mySet.standardize(128,64)
resized.save('cat.gif')
mySet.save('cat.gif') # save the cats as a gif
for cat in mySet: # iterate over the set of cats
    cat.binarize().show()
```

Getting at a pixel



- SimpleCV treats images as two dimensional arrays of color value tuples.
- Each tuple holds three values (Red,Green,Blue).
- Pixels start in the top left corner at zero.

Pixel Manipulation Example

Example (Getting at those pixels)

```
img = Image('helloworld.jpg')
c = img[0,0] # get a pixel
print c
print img.getPixel(0,0) # get another way
test = img[200:300,200:300]
test.show() # the result is an image
test = img[50:,:]
test.show() # again using slice
test = img[0:5,0:5]
print test.getNumpy() # get the raw values
img[0:105,0:105] = Color.RED
img.show() # Another way
test = img.getNumpy() #RIGHT!
test[0:105,0:105] = Color.RED
img2 = Image(test)
img2.show()
```

Getting at a pixel

- Images are read only. To write a pixel directly you need create a new image. Usually this happens in numpy
- Images support the list slice notation but return images.
- To get at the raw pixel values use **getNumpy()** and **getGrayNumpy()**
- Numpy values can be accessed using slices the first parameter is x, the second is y, and the third is the channel in RGB order. For example $npimg[x][y][0]$.
- The **Image.width** and **Image.height** member variables can help you find your way.

Another Pixel Manipulation Example

Example (Fancy Manipulations)

```
img = Image('helloworld.jpg')
gray = img.getGrayNumpy() # get the gray scale np image
colored = img.getNumpy()  # and the colored one
print (img.width,img.height) #tell us the image size
colored[0:20,:] = Color.BLUE # set the left side to blue
colored[:,0:20] = Color.GREEN # set the top row to green
colored[40:80,40:80]      \quad
     \includegraphics[width=0.4\linewidth]{JanEricBook.jpg}
= Color.YELLOW # make a yellow square
x,y = np.where(gray>230) # find bright pixels > 230
for xf,yf in zip(x,y): # for each of those
    colored[xf][yf] = Color.RED #make them red
img2 = Image(colored) # create an image
img2.show() # and show it
# now set the whole blue channel to 255
colored[:,:,2] = 255
# and show us that
img3 = Image(colored)
img3.show()
```

Getting at a pixel

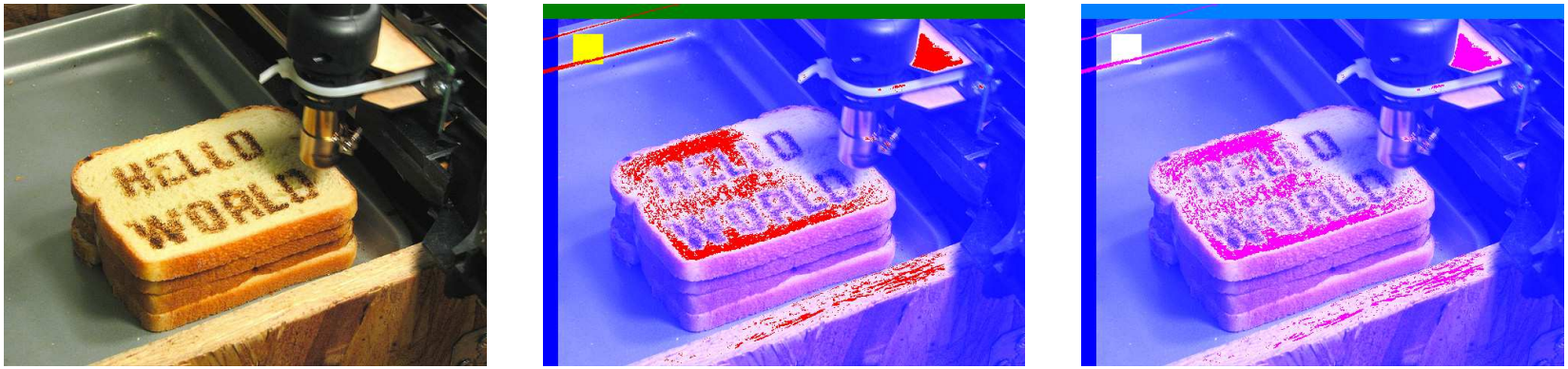


Figure: Not bad for less than 20 lines of code.

Cropping, Scaling, Rotating, etc

It is helpful to think of some image processing like using an image editor, like GIMP, paint, or that other one Adobe makes. Here are a few basic image operations you can do in SimpleCV.

- **Image.crop**. Does what it says on the tin. We refer to crop areas by there top left corner x and y plus the width height.
- **Image.scale** scales the image proportionally while **Image.resize** resizes the image to the desired size.
  - **Image.resize** is smart, if you tell it a width or height it will infer the other parameter from the aspect ratio.
  - **Image.scale** uses a proportionality. So passing it value of two will double the image size. Make sure to check out the interpolation method.

Rotating

- **Image.rotate** takes and angle in degrees.
- Rotate has a parameter called fixed. If fixed is set to false, SimpleCV will create a new image that matches the rotated image size. Otherwise we stick the rotated image in data in a similarly sized canvas.
- It is possible to pick the x,y position of the rotation. The default is the center of the image.
- You can also scale an image while rotating.
- In a pinch you can use **Image.flipHorizontal** and **Image.flipVertical**.
- Be aware **FLIPPING != ROTATION**

Blit

The **Image.blit** function gets its name from the old computer graphics term "bit block image transfer". Really it means just copy and paste another image onto another image and return the result.

- Blit takes in another image and a position where you want to put it.
- That position can be negative with respect to the destination image. For example (-10,-10) would put the source image over the top left of the destination image.
- You can toss blit a binary mask, an alpha value (that is transparency) or a an alpha mask (a grayscale image that has an alpha mask per pixel).

Let's play!

Example (Basic Manipulations)

```
img = Image('tricky.jpg')
face = img.crop(150,190,309-150,333-190)
img.show() # show the source
face.show() # show the cropped image
face.rotate(45).show() # basic rotation
face.rotate(45,fixed=False).show()
face.scale(0.5).show()
face.scale(width=int(face.width/2.0)) # basic scaling
face.flipHorizontal().show() # flipping
test1 = img.blit(face.flipHorizontal(),pos=(150,190))
test1.show() # now let's have fun with blitting
test2 = img.blit(face.flipHorizontal(),pos=(150,190),alpha=0.5)
test2.show()
mask = Image((face.width,face.height))
# Here we are just drawing a white circle on a black background
mask.drawCircle((face.width/2,face.height/2),70,color=Color.WHITE, thickness=-1)
mask = mask.applyLayers()
mask.show()
test3 = img.blit(face.flipHorizontal(),pos=(150,190),mask=mask)
test3.show()
face.binarize().blur().show()
test4 = img.blit(face.flipHorizontal(),pos=(150,190),alphaMask=face.binarize().blur())
test4.show()
```

The Blit Progression of President Dick Nixon




Figure: Various blitting

Can you our subject a smaller face or a slightly slanted face?

A few more basics

- **Image.blur** - A basic Gaussian blur.
- **Image.smooth** - A variety of smoothing filters. Some good for removing camera noise.
- **Image.toGray** - Convert a color image to a gray scale image.
- **Image.threshold** - Take an image and set all off the pixels that have a grayscale value above the threshold to white, and everything else to black.
- **Image.invert** - Swap the lightest and darkest values, like a photo negative.

SimpleCV treats on image rendering as a separate layer.

- Each image has a stack of transparent drawing layers.
- Every time you call a draw command we draw on the transparent layer.
- This is layer is rendered when you call the **Image.show** command or save it to a display.
- **Image.save** does not save the drawing layer by default.
- Use the **Image.applyLayers** command to return a new image with the layer applied.
- The **Image.clearLayers** command removes the layers.

Accessing the drawing layer.

- **Image.dl** method returns the drawing layer that has more advanced rendering features.
- The drawinglayer is really a list of layers if you need them.
- You can set the layer alpha value.
- Allows for more fine grained control.
- Use sprite for non-destructive blit function.
- Use the **Color** class for colors or set your own.

Basic Drawing Primitives

- **Image.drawCircle** - Draw a circle based on center and radius.
- **Image.drawRectanle** - Draw a rectangle on top left corner and width and height.
- **Image.drawRotatedRectangle** - Draw rectangle from rotated points.
- **Image.drawText** - Draw a line of text at a position
- **Image.drawLine** - Draw a line based on two (x,y) points.
- **Image.drawPoints** - Put a circle around points of interest.

Go crazy with drawing.

Example (Basic Manipulations)

```
img = Image('tricky.jpg')
poly = [(0,0),(30,50),(123,213),(234,234),(333,434)]
img.dl().polygon(poly,color=Color.GREEN,filled=True)
for i in xrange(30,90,9):
    poly2 = [(p[0]+i,p[1]+i) for p in poly]
    img.dl().polygon(poly2,color=Color.getRandom(),filled=True,alpha=128)
img.drawRectangle(130,190,200,50,color=Color.BLACK,width=-1,alpha=128)
img.drawCircle((170,220),20,color=Color.WHITE)
img.drawCircle((230,220),20,color=Color.WHITE)
img.drawLine((0,0),(100,100),color=Color.PUCE,thickness=3)
img.drawLine((300,200),(300,200),color=Color.PUCE,thickness=3)
img.drawLine((45,300),(300,45),color=Color.PUCE,thickness=3)
img.drawText('HELLO WORLD!',30,30,fontsize=60)
img.show()
img.applyLayers().save('crazynixon.png')
img.clearLayers()
img.show()
```

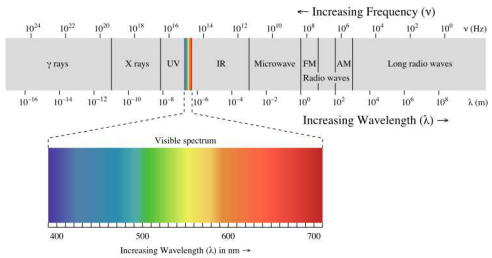
Crazy Drawing Stuff.

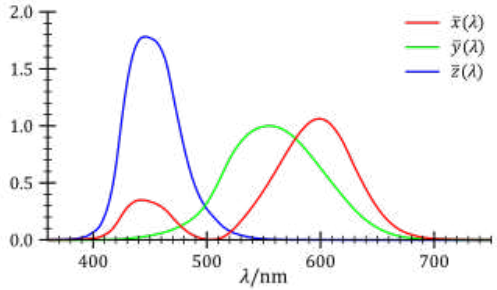
Figure: Various drawing things.

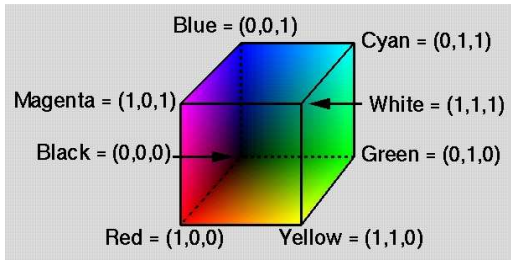
Let's Talk about color, color spaces, and much more.

Computer vision is all about moving from a vast amount of information to small result as quickly as possible. Working in grayscale, or black and white images can speed things up dramatically.

Color is surprisingly hard to represent

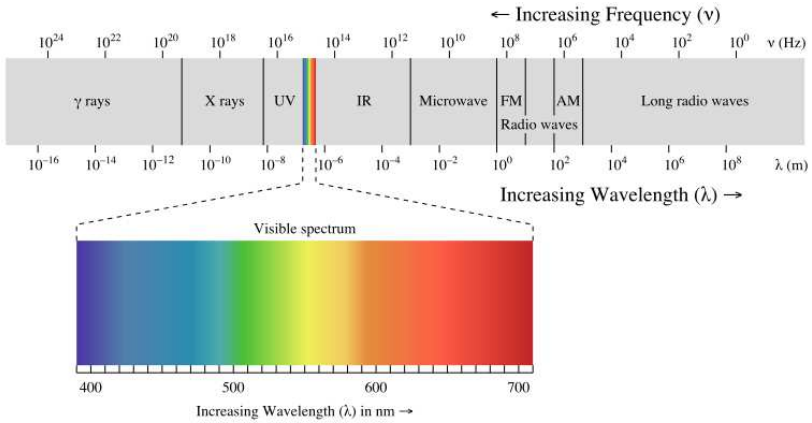






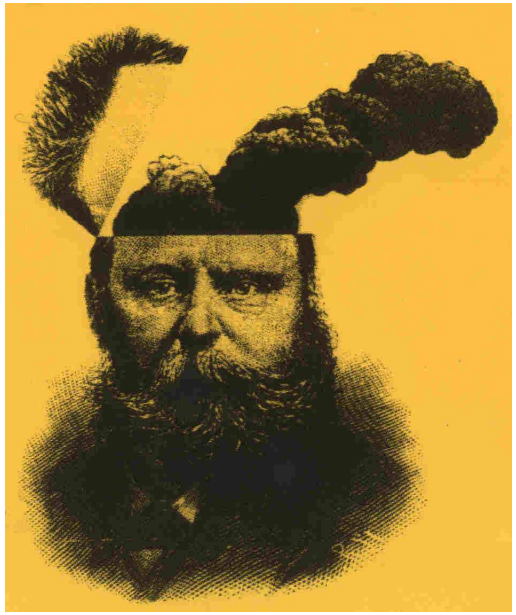
We start with visible light, which bounces around, does funky stuff, and then enters our eye or camera. Our eyes and cameras have a response function that does an imperfect job of sampling the parts of the spectrum. We then take those samples and try to map them onto finite color space like RGB.

An illustration of why color is hard.



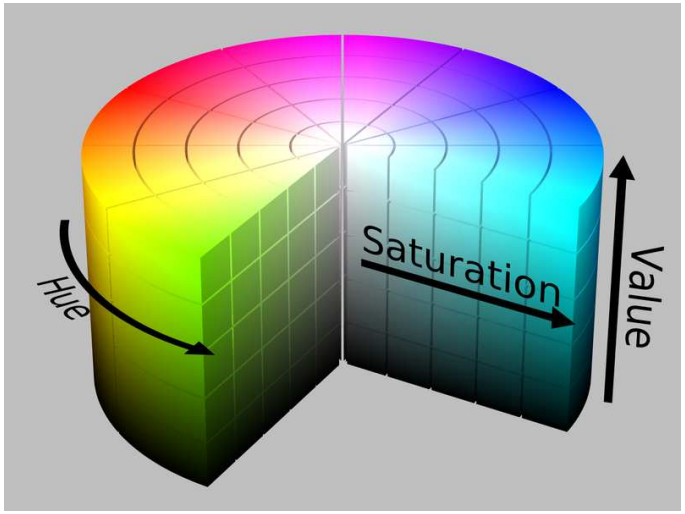
Point to where this magenta flower lives on the visible light spectrum.

MIND BLOWN!



Magenta doesn't exist in nature. It is trick our brains and cameras play on us. It exists because we sample the spectrum and try to recombine the samples. We wrap the visible spectrum around.

To manage this problem we use color spaces.



- To deal with this problem we use color spaces.
- Most images are in the RGB, BGR, or gray scale color spaces.
- Sometimes it is helpful to use the hue, saturation, and value (HSV) color space. In HSV you have a
  - Hue, or pure color (from 0 to 180)
  - Saturation that tells us how far from white our color is
  - Value that tells us how dark the color is.

How do I work with Color in SimpleCV

- The **Image.toHSV()** function will convert your image, while the **Image.toBGR()** function will return it back to the original.
- We also have **Image.toGray()** and **Image.toRGB()** and whole bunch more.
- The **Image.huePeaks** function can help you guess what hues are in your image.
- The **Image.hueDistance** function can then help you look for stuff that is the hue you want.
- The **Image.hueHistogram** function will let you visualize the results.

Finding colors

Example (Using Hue)

```
import pylab as plt # import pylab for plotting
img = Image("flower.jpg")
img.show()
peaks = img.huePeaks() # find the hues
print peaks
hist = img.hueHistogram() # get the histogram too
plt.plot(hist) # plot the histogram
plt.draw() # show it to us
# show how far away from the hue we are black is closer
# white is farther away from our hue.
hdist = img.hueDistance(peaks[3][0])
# create a black and white version of our flower
binary = img.hueDistance(peaks[3][0]).invert().threshold(220)
hdist.show()
binary.show()
hdist.save('hflower.png')
binary.save('bflower.png')
```

Color Finding Results

Figure: Finding an object from colors

How do I work **without** Color in SimpleCV

Color is not so important when all we care about are light or dark things.

- We can use the **Image.toGray** method to create gray images. Gray pixels only go from values of 0 to 255
- Sometimes it is helpful to adjust the contrast for us to find objects.
  - If the gray values in our image only go from say [50,150] we can us the **Image.equalize** method to interpolate them to values between [0,255].
  - Other times we want to enhance a certain space of the gray spectrum. We can use the **Image.stretch** function.
- **Image.histogram** can help us figure out what gray values are present.
- **Image.maxVal** and **Image.minVal** can help us determine the range of intensities and where to set our thresholds.

Shades of Gray

Example (Modify Grayscale Images)

```
img = Image('penguins.jpg')
img = img.toGray()
eImg = img.crop(180,400,100,100)
img.drawRectangle(180,400,100,100)
plt.plot(eImg.histogram(), '-r')
plt.plot(eImg.equalize().histogram(),'-b')
plt.show()
img.show()
eImg.show()
eImg.equalize().show()
max = eImg.maxValue()
min = eImg.minValue()
print [max, min]
streched = img.stretch(min,max)
streched.show()
```

Seeing in Black and White - Binary Images

Very often we want to just isolate a certain area in an image and mark it. To do this we use binary images, also called masks. The general process of creating a binary image is called segmentation. Often we call the white areas foreground and the black areas background.

- We can use the**Image.threshold** method to create a binary image.
  - Threshold will set gray values below the threshold to black and areas above the threshold to white.
  - SimpleCV will automatically convert color images to grayscale images for a threshold.
- The **Image.binarize** method will also create a binary image.
  - Binarize uses Otsu's method which changes the threshold automatically to improve performance.
  - There are a lot of parameters to tweak so checkout the documentation.

Basic Binarization

Example (Let's find the parrot)

```
img = Image('parrot.jpg')
b = img.binarize().invert() # automatic
t1 = img.threshold(50) # too low
t2 = img.threshold(128) # okay
t3 = img.threshold(200) # too high
b.show()
t1.show()
t2.show()
t3.show()
b.save('bparrot.png')
t1.save('t1parrot.png')
t2.save('t2parrot.png')
t2.save('t3parrot.png')
```

Binzarization Results

Left to right, source image, binarize using Otsu's method, threshold = 50, threshold=128, threshold=200

Morphological Operations - Fixing Binarized Images

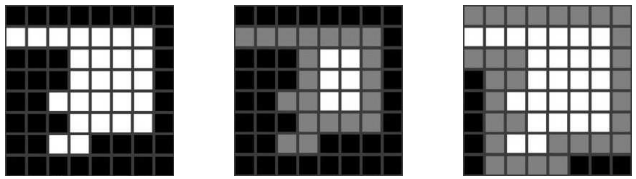


Figure: Source image, source after erosion, source after dilation.

We can clean up binary images using basic morphology operations.

- We can use the**Image.dilate** to enlarges our white areasand connects them.
  - Basically if a pixel touches a white pixel we set it to white.
  - Can be used repeatedly. Works in color too.
- The **Image.erode** method shrinks our white areas image.
  - If a white pixel touches a black pixel we set it to black.
  - Can be used repeatedly.
- **Image.morphOpen** and **Image.morphClose** do similar things but are used to open and close regions.

Fixing Binarized Images II

There are other techniques we can use to fix things.

- **Image.floodFill** works just like the paint bucket tool in an image editor. It can get rid of areas.
- **Image.watershed** is a sophisticated technique that can really help.
- We can also logically and, or, xor, and nand images.
  - Logical and is just multiplication or **Image.logicalAND**
  - Logical or is just multiplication or **Image.logicalOR**
  - Logical xor is **Image.logicalXOR**
  - Logical nand is **Image.logicalNAND**
  - These all work in color and gray scale too.

Advanced Binarization

Example (Get a binary representation of the parrot.)

```
img = Image('parrot.jpg')
img.show()
binary = img.binarize().invert() # automatic
binary.show() # good but we're missing part of the beak
binary2 = img.hueDistance(80).invert().threshold(190)
binary2.show() # missing a different part of the beak
filled = binary.floodFill((0,img.height-1),color=Color.BLACK)
filled.show() # get rid of the branch in the corner
better = filled.logicalOR(binary2)
better.show() # combine the two.
eroded = better.erode()
eroded.show() # get rid of specs.
dilated = better.dilate()
dilated.show() # get rid of holes
watershed = img.watershed(dilated)
watershed.show() # let's see what this can do
```

Parrot Pipeline

Or pipeline, left to right, top to bottom

Finding Stuff

Now that we've got a lot of the basics down let's start doing stuff with images. Let's **find** interesting sets of **features** in our images.

Finding Stuff II

SimpleCV allows you to quickly and easily find **features** in your images using a variety of methods that begin with the word **find**.

Features have a set of basic properties that makes manipulating them super easy.

The **Image.findXXXX** methods each return a **FeatureSet** which is a list of features.

A **FeatureSet** also allows you to look at aggregate information about features.

Things you can find in SimpleCV

- Blobs - binary objects
- Lines
- Corners
- Circles
- Haar Features
  - faces
  - eyes
  - mouths
  - much more.
- Barcodes
- Text
- Templates (example images)
- Keypoints (interesting areas)
- Keypoint Matches
- Motion (between images)
- Skintone Blobs

The Basic Mechanics

Once you have a **FeatureSet** you can use iteration either in a loop or using python list comprehensions to filter your features to get what you want.

- **Feature.draw** Will draw the feature on the source image.
- **Feature.show** Will show the feature on the source image.
- **Feature.crop** Will return an image of just the feature.
- **Feature.meanColor** Will return the average color.
- The feature's width, height, and position can also be checked.

The Basic Mechanics II

- **Feature.area** gives the area of the feature.
- **FeatureSet.filter** Allows you to filter features on different criteria.
- **FeatureSet.distanceFrom** will return the distance of each feature from a point.
- **FeatureSet.area** returns the area of every feature as a list. By default we sort the feature set from smallest to largest.
- FeaturesSets also know the center points of the features and their corners.

Finding Blobs

Blob is a backronym for binary large object. They are basically anything that would be white in a binary image. You can find blobs in a variety of ways:

- **Image.findBlobs** will do a binarize and return the blobs it found.
- **Image.findBlobs** will use a binary image, also called a masks, to find blobs.
- **Image.findBlobsFromPalette** will find blobs with specific colors from a palette
- **Image.findBlobsFromWatershed** will find blobs using the watershed algorithm and a binary image you provide

Let's see an example.

Can we write some python to find the star cookie in the top right corner?

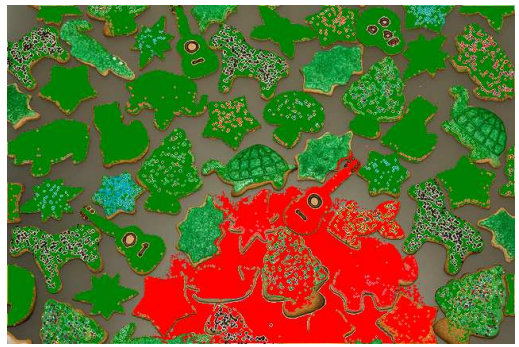
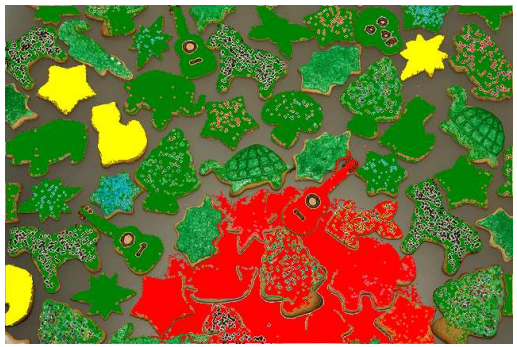
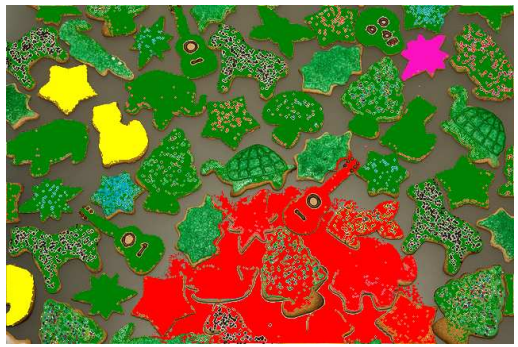
Sorting blobs

Example (Finding the yellow cookie in the corner.)

```
img = Image('cookies.jpg')
# find our cookes and draw them red, filled in
blobs = img.findBlobs(threshval=128, minsize=200)
blobs.draw(width=-1, color=Color.RED) #autocolor= True)
# find the mean and std blob areas
areaAvg = np.mean(blobs.area())
areaStd = np.std(blobs.area())
# filter the cookies by area and draw those green
lilcookies = blobs.filter(blobs.area() < areaAvg+2.5*areaStd  )
lilcookies.draw(width=-1,color=Color.GREEN)
# Now sort the cookies so the yellow ones are at at 0
lilcookies = lilcookies.sortColorDistance(color=Color.YELLOW)
lilcookies[0:4].draw(width=-1,color=Color.YELLOW)
# Now take our yellow cookies and see how
# far they are away from the top right corner
dists = lilcookies[0:4].distanceFrom((img.width,0))
# find the closest one to the corner
location = np.where(dists==np.min(dists))[0][0]
lilcookies[location].crop().show()
lilcookies[location].draw(width=-1,color=Color.HOTPINK)
img.show()
```

Cookie Sorting



Advanced Blobs

Blobs have a low other neat features.

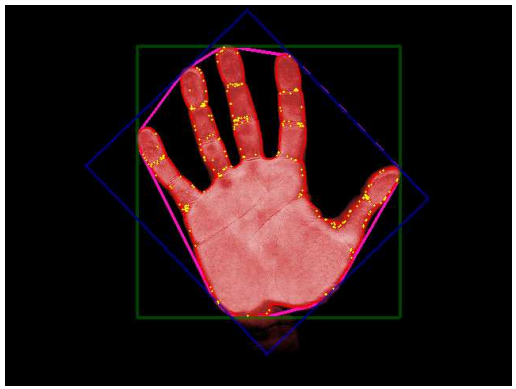
- **Blob.minRect** The rotated rectangle that best encloses the blobs.
- **Blob.contour** A list of (x,y) tuples and other lists of (x,y) tuples that describe the outer contour.
- **Blob.hull** The contour of the blob if you were to stretch a rubber band around it. This eliminates what we call concavities.
- *Blob.mHoleContour* gives a list of the holes and "islands" in the holes of the blob. This is a list of list that may have lists inside of them.

Advanced Blobs II

Blobs have a lot of other neat features.

- **Blob.drawRect** Draw the square region around the blob.
- **Blob.drawMinRect** Draw the minimum bounding rectangle for each blob.
- **Blob.drawOutline** The outline of the blob without the holes. may have lists inside of them.
- **Blob.drawHoles** Draw just the holes.
- **Blob.drawHull** Draw just the blob's convex hull. (The rubber band version.)
- *These are order dependent! They can overwrite each other*

Let's see an example.



Example

```
img = Image('hand.jpg')
blobs = img.findBlobs(minsize=200)
blob = blobs[0]
blob.drawHull(color=Color.HOTPINK,width=4)
blob.draw(color=Color.RED,width=2)
blob.drawRect(color=Color.GREEN, width=4)
blob.drawMinRect(color=Color.BLUE, width=4)
blob.drawHoles(color=Color.YELLOW, width=-1)
img.show()
img.applyLayers().save('handblob.png')
```

Advanced Blobs III - Masks

Sometimes we want to work with binary image that comes from a blob.

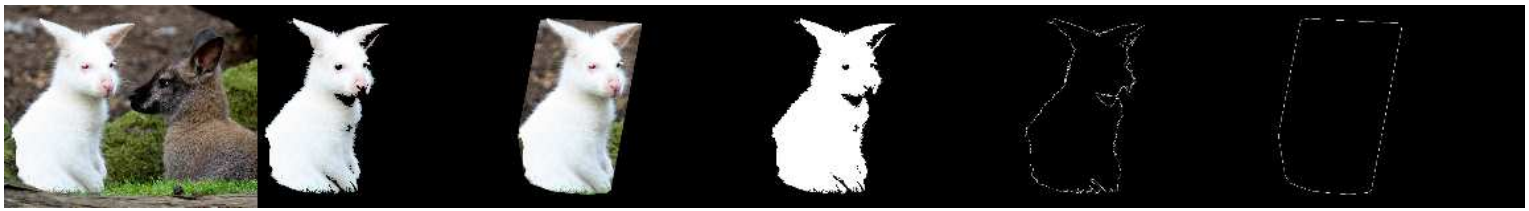
- **Blob.blobImage** An image of just the blob region.
- **Blob.hullImage** An image of just the blob's hull region.
- **Blob.blobMask** A binary image of just the blob.
- **Blob.hullMask** A binary image of the blob's convex hull.
- **Blob.getFullMaskedImage** Get the original image with everything black except for the blob. There is also a convex hull version.
- **Blob.getFullMask** Get the original sized as a binary mask of the blob. Also has a hull variant.
- **Blob.getEdgeImage** return an image of just the blobs outer edge. Also has full and hull variants.

Masks, Edges, etc

Example

```
img = Image('albino.jpg')
img.show()
blobs = img.findBlobs(minsize=200)
blob = blobs[-1]
src = blob.mImg
src = src.sideBySide(blob.blobImage())
src = src.sideBySide(blob.hullImage())
src = src.sideBySide(blob.blobMask())
src = src.sideBySide(blob.hullMask())
src = src.sideBySide(blob.getEdgeImage())
src = src.sideBySide(blob.getHullEdgeImage())
src = src.scale(0.5)
src.show()
src.save('albinoblob.png')
big = img
big = big.sideBySide(blob.getFullMaskedImage())
big = big.sideBySide(blob.getFullHullMaskedImage())
big = big.sideBySide(blob.getFullMask())
big = big.sideBySide(blob.getFullEdgeImage())
big = big.sideBySide(blob.getFullHullEdgeImage())
big = big.scale(0.3)
big.show()
big.save('albinoimgs.png')
```

Comparing Mask Options



Advanced Blobs IV - Shape

We can use shape information to compare blobs

- **Blob.area** The total number of pixels in the blob.
- **Blob.perimeter** The length of the outside of the blob.
- ProTip the area/perimeter is a nice metric for smoothness.
- **Blob.angle** The angle between the minimum rectangle and horizontal in degrees.
- **Blob.centroid** The center of mass, not the center of the bounding box.
- **Blob.rotate** Rotate a blob a certain number of degrees. This changes the blobs internal state.

Advanced Blobs V - Shape

We can use shape information to compare blobs

- **Blob.circleDistance** and **Blob.isCircle** help to find circular things.
- **Blob.rectangleDistance** and **Blob.isRectangle** help to find rectangular things
- **Blob.isSquare** returns true if a blob is squareish.
- *Blob.mHu* A list of the seven Hu moments. These are like moments of inertia in physics. Hu moments don't care about rotation, and they are really helpful for matching blobs
- **Blob.match** returns a match value against a particular blob based on shape using Hu moments.

Matching blobs

Example (Try to find the horse cookies given a template.)

```
img = Image('cookies.jpg')
# find our cookes and draw them red, filled in
blobs = img.findBlobs(threshval=128, minsize=200)
blobs.draw(width=-1, color=Color.RED) #autocolor= True)
# find the mean and std blob areas
areaAvg = np.mean(blobs.area())
areaStd = np.std(blobs.area())
# filter the cookies by area and draw those green
lilcookies = blobs.filter(blobs.area() < areaAvg+2.5*areaStd  )
lilcookies.draw(width=-1,color=Color.GREEN)
# Now sort the cookies so the yellow ones are at at 0
lilcookies = lilcookies.sortColorDistance(color=Color.YELLOW)
lilcookies[0:4].draw(width=-1,color=Color.YELLOW)
# Now take our yellow cookies and see how
# far they are away from the top right corner
dists = lilcookies[0:4].distanceFrom((img.width,0))
# find the closest one to the corner
location = np.where(dists==np.min(dists))[0][0]
lilcookies[location].crop().show()
lilcookies[location].draw(width=-1,color=Color.HOTPINK)
img.show()
```

Cookie Matching Results

Finding Lines

SimpleCV makes it easy to find straight lines.

- **Image.edges** creates an image of all the lines in the image using the Canny edge detector.
- There are two thresholds that you can tweak to improve performance.
- **Image.findLines** will apply the Canny edge detector and then use the Hough transform to find the straight lines.
- **Image.findLines** returns a FeatureSet of Line features that you can then filter.

The Line Feature

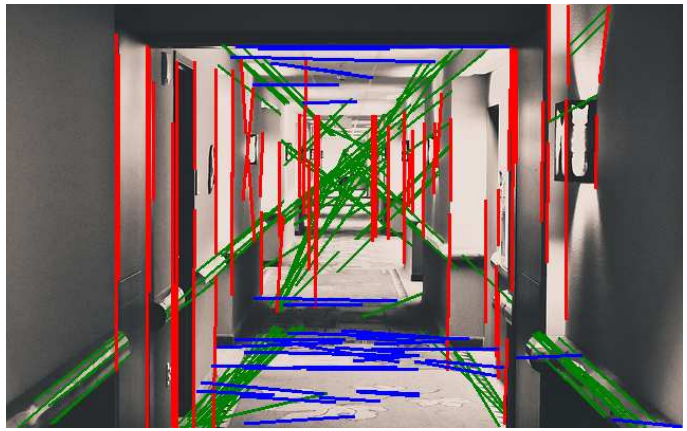
- The line feature can be found in Detection.py
- Each line knows its length, its average color, its angle.
- **Line.findIntersection** can be used to find line intersections.
- You can also test for parallel and perpendicular lines

Line Finding Example

Example

```
img = Image('hallway.jpg')
img.show()
img.edges().show()
img.edges().save('hallwayedges.png')
lines = img.findLines() # get the lines
# return only long ones
lines = lines.filter(lines.length() > 50 )
lines.show(width=3)
# now find the horz and vert lines by
# filtering on the angle.
vertical = lines.filter((np.abs(lines.angle()) > 80))
horizontal = lines.filter(np.abs(lines.angle()) < 10 )
vertical.draw(width=3,color=Color.RED)
horizontal.draw(width=3,color=Color.BLUE)
img.show()
img.applyLayers().save('hallwaylines.png')
```

Line find results.



Finding Circles

SimpleCV makes it easy to find circles too.

- Just like before we use the edge image.
- **Image.findCircle** returns a FeatureSet of Circle features that you can then filter.
- There are three thresholds that you can tweak to improve performance. One for the edges, one for the circle detector, and one for the circle spacing.
- Circle features have all of the usual features like mean color along with the ones you would expect like radius, diameter, etc.

Circle Finding Example

Example

```
img = Image('clock.jpg')
img.show()
temp = img.binarize()
circles = temp.findCircle(thresh=200)
circles.show(width=3)
temp.applyLayers().save('circleedges.png')
circles = circles.sortArea()
circles[-1].crop().show()
circles[-1].crop().save('clockface.png')
```

Circle finding results.

Template Matching

Sometimes you have an image that you want to use as a "template" and you want to match it to something in your image.

- **Image.findTemplate** returns a FeatureSet of template features that you can then filter.
- Templates can handle a little bit of rotation, illumination change, and skew, but not much.
- you can try slightly rotating your template, or using the edge image of something like the **Image.sobel** operation to improve performance.
- The performance of the matching can be tuned using different matching methods and by tweaking the threshold.

Template Finding Example

Example

```
img = Image('coins1.jpg')
mask = img.threshold(130).invert()
blobs = img.findBlobsFromMask(mask, minsize=200)
blobs.show(width=-1)
img.clearLayers()
blobs[-5].crop().show()
template = blobs[-5].crop()
template.save('template1.png')
found = img.findTemplate(template,threshold=2.2)
found.draw(color=Color.RED)
template2 = blobs[5].crop()
template2.save('template2.png')
template2.show()
found2 = img.findTemplate(blobs[5].crop(),threshold=4.15)
found2.draw(color=Color.BLUE)
found2.show()
img.save('foundtemplates.png')
```

Template finding results.

Motion Finding

Another common task is looking for motion. If you just want to detect change looking at the difference between images usually suffices. However sometimes you want to know more.

- **Image.findMotion** returns a FeatureSet of motion values between two images.
- This method takes in a second image and then reports the motion. The image sizes must match.
- SimpleCV uses an algorithm called optical flow of which there are several variants. You can pick the one that works best for you.
- There is one parameter to tweak, called window, which is the sample window size. Faster movement requires larger windows.

Motion Finding II

- The images should have a reasonable amount of edges, and be fairly close together in time.
- Motion will be most obvious around strong edges, less so around plain areas, but some false positives tend to happen.
- By default the rendered motion is scaled to unit length, but the actual values vary considerably.
- Be aware that there is motion from the camera, and motion from objects moving.
- Works a lot better on a live camera.
- The algorithms can be computationally beefy, scale images to get better results.

Motion Finding Example I

Example

```
img1 = Image('toys1.jpg')
img1 = img1.scale(0.5)
img1.show()
img2 = Image('toys2.jpg')
img2 = img2.scale(0.5)
img2.show()
motion = img2.findMotion(img1,method="BM",window=27)
motion.show(width=3)
img2.clearLayers()
mags = [m.magnitude() for m in motion]
bigMove = motion.filter( mags > np.mean(mags))
bigMove.show(color=Color.RED, width=3)
bigMove.applyLayers().save('motion.png')
```

Motion finding results.

Motion Finding Example II

This works a lot better as a live example.

Example (MotionExample.py)

```
1  from SimpleCV import *
2  scaler = 0.5
3  cam = Camera()
4  disp = Display((640,480))
5  last = cam.getImage().scale(scaler)
6  sz = last.width/10
7  while disp.isNotDone():
8      img = cam.getImage().scale(scaler)
9      motion = img.findMotion(last,sz,method='HS')
10     motion.draw(color=Color.RED,width=3)
11     img.save(disp)
12     last = img
```

Finding Text and Reading Barcodes

Text and barcode reading are optional libraries that we've added hooks for.

- **Image.findBarcodes** uses either lib zxing or zlib.
  - Each has its advantages and disadvantages. Generally zxing works better.
  - Returns a barcode feature with the location and the data.
- **Image.readText** uses python bindings to the tesseract library.
  - Does not work in the SimpleCV ipython notebook, but we're working on it.
  - Translation to string often leaves lots of spare characters. Formatting is an issue.

Reading a QR Code

Example

```
qrimg = Image('qrcode.png')
qrimg.show()
bc = qrimg.findBarcode()
bc.show(width=3)
print bc[0].data
qrimg.drawText(str(bc[0].data),10,10, \\
               color=Color.RED,fontsize=24)
qrimg.show()
img = Image('textcard.jpg')
img.show()
print img.readText()
```

Face Finding

There are built in utilities in SimpleCV for finding faces and parts of faces.

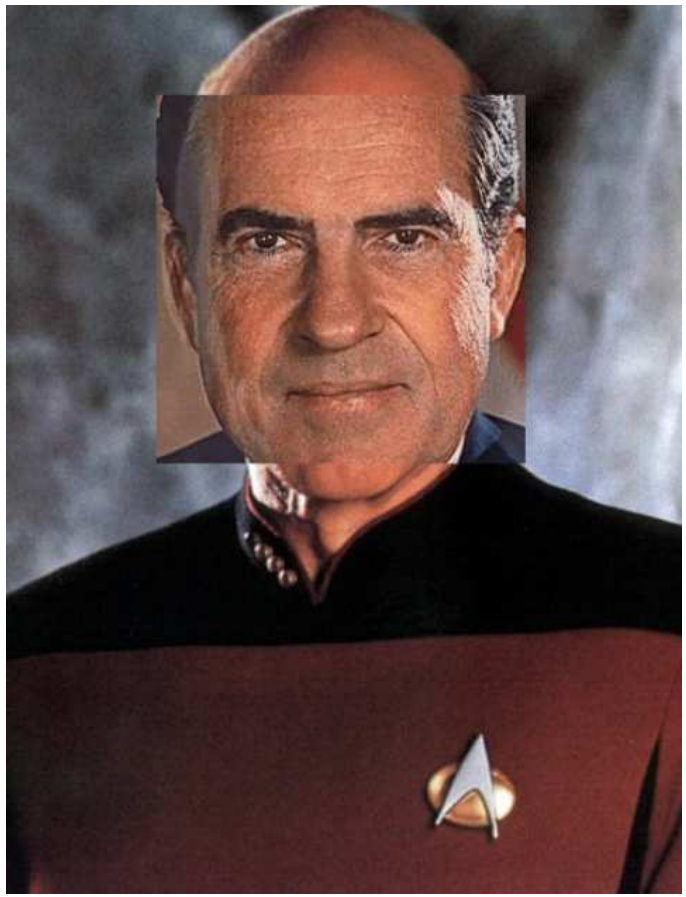
- These parts of the face are found using what is called a Haar Cascade.
- Lots of data and research went into developing these tools.
- **Image.findHaarFeatures** returns a FeatureSet of features corresponding to the cascade you pass it.
- **Image.listHaarFeatures** will list all of the ones available, the default is front faces.
- Not perfect, can't get all faces all the time. Sometimes it finds false positive.

Fun With Faces Example 1

Example

```
def swap(img1,f1,img2,f2):
    print f1.width
    print f1.height
    f1mask = f1.crop().resize(w=f1.width(),h=f1.height())
    f2mask = f2.crop().resize(w=f1.width(),h=f1.height())
    out1 = img1.blit(f2mask,f1.topLeftCorner(),alpha=0.8)
    out2 = img2.blit(f1mask,f2.topLeftCorner(),alpha=0.8)
    return out1,out2
img1 = Image('tricky.jpg')
img2 = Image('picard.jpg')
img1.show()
img2.show()
face1 = img1.findHaarFeatures('face.xml')
face1.show()
face2 = img2.findHaarFeatures('face.xml')
face2.show()
img3, img4 = swap(img1,face1[0],img2,face2[0])
img3.show()
img3.save('swap1.png')
img4.show()
img4.save('swap2.png')
```

Leaders of the free world.

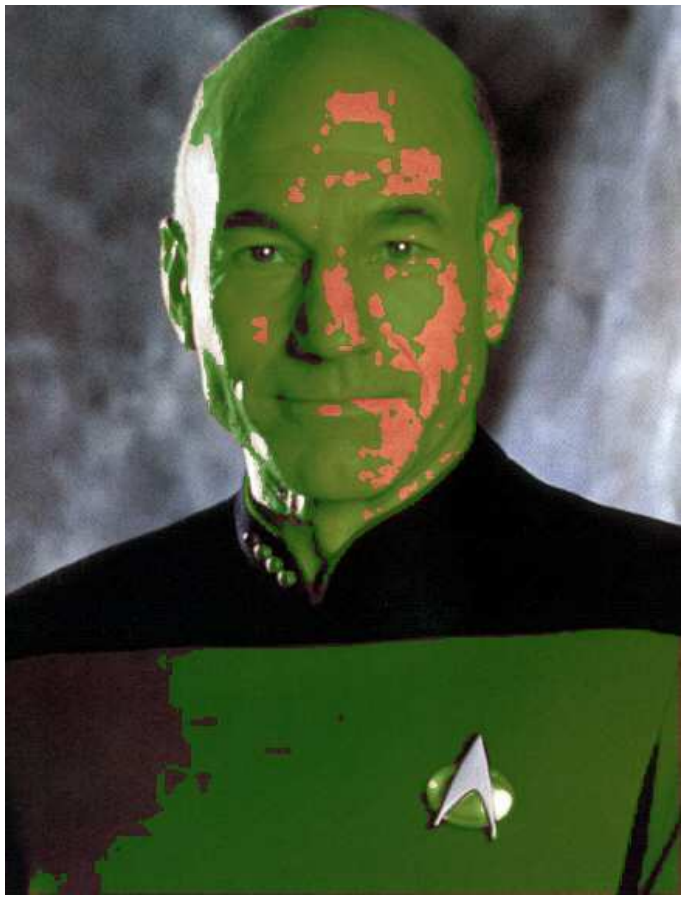


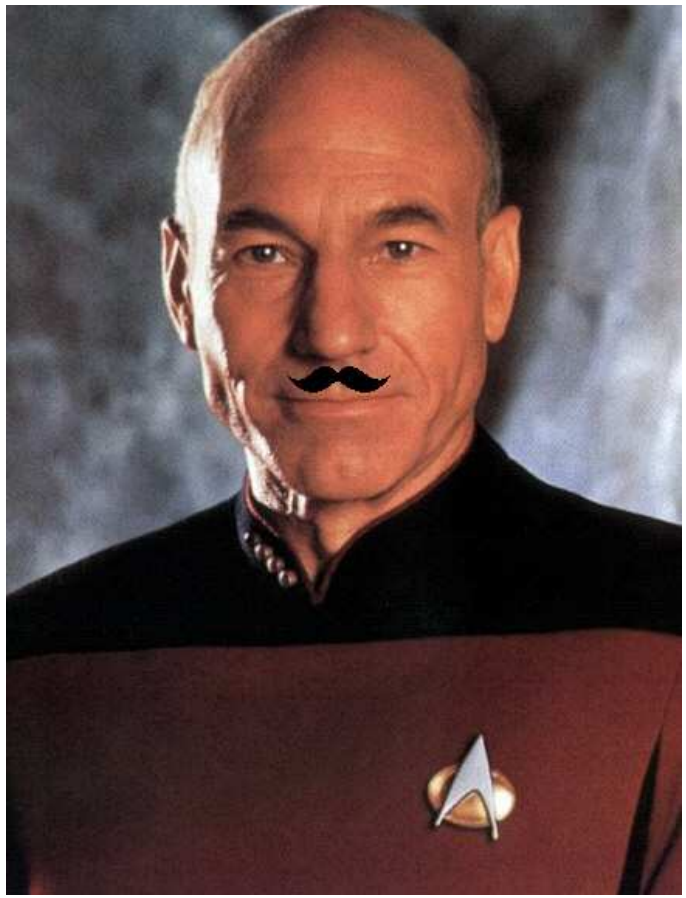
Fun With Faces Example 2

Example

```
img = Image('picard.jpg')
blobs = img.findSkintoneBlobs()
print len(blobs)
for b in blobs:
    b.draw(color=Color.GREEN,alpha=128,width=-1)
img.show()
img.applyLayers().save('alien.png')
img.listHaarFeatures()
mouth = img.findHaarFeatures('mouth.xml')
mustache = Image('stash.png')
mustache = mustache.resize(mouth[0].width())
tl = mouth[0].topLeftCorner()
pos = (tl[0],tl[1]-10)
img = img.blit(mustache,pos,mask=mustache.invert())
img.show()
img.save('psstash.png')
```

Leaders of the free world.





Introduction To ML

SimpleCV has a number of utilities for doing supervised machine learning.

- Supervised machine learning is where we show an algorithm examples of what we want to classify and it tunes itself to optimally classify objects.
- Usually this type of machine learning is used for:
  - Tell one thing from another. E.g. cats vs dogs.
  - Tell multiple things from each other E.g. cats vs dogs vs ponies.
  - Tell if one or more things exist in an image.

The process.

- Collect some data – either manually, by scraping the web, or using an existing data set.
- Chop the data into subsets. The most basic subset is to divide the data into two parts: a test set and a train set.
- Extract features from the data and try a variety of classification techniques.
  - This is part art and part science. Each classifier has different properties, and so does each kind of feature.
  - There are techniques for determining what features are more salient than others.
- Train the classifiers and apply them to new data. Pick the one that has the best results.
- Deploy the classifier.

The Mechanics

- **FeatureExtractors** can used to extract common "feature vectors" from images as lists.
  - **HueHistogramFeatureExtractor** - returns a histogram of hues.
  - **EdgeHistogramFeatureExtractor** - The angles of the line in the image.
  - **HaarLikeFeatureExtractor** - Rough measures of symmetry and shape.
  - **BOFFeatureExtracor** - Little tiny edge gradient features.
  - **MorphologyFeatureExtractor** - Blob Hu moments, length, width, aspect ratio, perimeter, etc.
  - Many more – write your own!

The Mechanics II

- **Classifiers** Can then be used to classify images based on these "feature vectors"
  - **KNNClassifier** - Return the class of the feature set in the training data that is closest to the input vector.
  - **SVMClassifier** - Support vector machine, find a mathematical function that best separates our classes.
  - **NaiveBayesClassifier** weight the probability of the class appearing along with the probability of a feature appearing.
  - **TreeClassifier** - Basically a big long list if/else statements to reach a decision. Lots of different flavors of trees and forests (lots of trees).

The Mechanics III

- Training and testing machine learning is easiest if you save your data to file, usually sorted into directories by class.
- **TurkingModule** can help you manually sift through your data and save it.
- **ConfusionMatrix** class can be used to help you figure out how well your classifier works. Prints out what classes are confused with what other classes.
- Use pickle to save classifiers.

I'll have what he's having

Data has already been downloaded from Google image search and saved in the data directory.

Example

```
hhfe = HueHistogramFeatureExtractor(10) #look for hue
ehfe = EdgeHistogramFeatureExtractor(10) # look at edge orientation
# look at the basic shape
haarfe = HaarLikeFeatureExtractor(fname="../../../SimpleCV/SimpleCV/Features/haar.txt")
extractors = [hhfe,ehfe,haarfe] # put these all together
svm = SVMClassifier(extractors) # try an svm, default is an RBF kernel function
tree = TreeClassifier(extractors) # also try a decision tree
trainPaths = ['./data/wine/train/','./data/beer/train/','./data/whiskey/train']
testPaths = ['./data/wine/test/','./data/beer/test/','./data/whiskey/test/']
# define the names of our classes
classes = ['wine','beer','whiskey']
# train the data
print svm.train(trainPaths,classes,verbose=False)
print tree.train(trainPaths,classes,verbose=False)
print "---------------------------------------"
# now run it against our test data.
print svm.test(testPaths,classes,verbose=False)
print tree.test(testPaths,classes,verbose=False)
```

Making Sense of the Results

Example

```
# SVM training results -- good
# percent correct / percent incorrect / confusion matrix
[100.0, 0.0, [[15.0, 0.0, 0.0], [0.0, 15.0, 0.0], [0.0, 0.0, 15.0]]]
# The basic decision tree
Angle_feature4x3_12 (<15.000, 15.000, 15.000>)
: <=1208958.500
   Angle_feature3x2_5 (<8.000, 15.000, 15.000>)
# <---- SNIP ---->
: >1208958.500 --> wine (<7.000, 0.000, 0.000>)
# Tree training results 91 percent correct 9 incorrect
# two wines were classified as whiskey, one as a beer
# one whiskey was labeled wine
[91.11111111111111, 8.88888888888889, [[15.0, 0.0, 0.0], [2.0, 12.0, 1.0], [0.0, 1.0, 14.0]]]
----------------------------------------
# The results on all new data for SVM
[77.77777777777779, 22.22222222222222, [[10.0, 0.0, 5.0], [0.0, 10.0, 5.0], [0.0, 0.0, 15.0]]]
# The results on all new data for our Tree
[80.0, 20.0, [[14.0, 0.0, 1.0], [3.0, 11.0, 1.0], [2.0, 2.0, 11.0]]]
```

Visualizing the Results

Example

linenos

```
import random
test = ImageSet()
for p in trainPaths: # load the data
    test += ImageSet(p)
random.shuffle(test) # shuffle it
test = test[0:10] # pick ten off the top
i = 0
for t in test:
    className = tree.classify(t) # classify them
    t.drawText(className,10,10,fontsize=80,color=Color.RED)
    fname = "./timgs/classification"+str(i)+".png"
    t.applyLayers().resize(w=128).save(fname)
    i = i + 1
test.show()
```

Classification Results

The End – GO HAVE FUN!