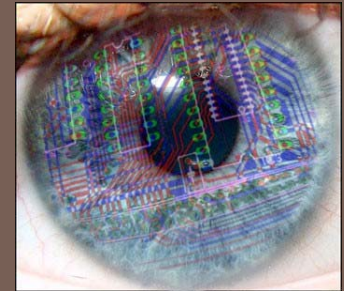


COMPUTER VISION USING SIMPLECV AND THE RASPBERRY PI

Cuauhtemoc Carbajal
ITESM CEM



Reference:

Practical Computer Vision with SimpleCV - Demaagd (2012)

Enabling Computers To See

2

- SimpleCV is an open source framework for building computer vision applications.
- With it, you get access to several high-powered computer vision libraries such as OpenCV – without having to first learn about bit depths, file formats, color spaces, buffer management, eigenvalues, or matrix versus bitmap storage.
- **This is computer vision made easy.**

SimpleCV is an open source framework

3

- ❑ It is a collection of libraries and software that you can use to develop vision applications.
- ❑ It lets you work with the images or video streams that come from webcams, Kinects, FireWire and IP cameras, or mobile phones.
- ❑ It's helps you build software to make your various technologies not only see the world, but understand it too.
- ❑ SimpleCV is written in Python, and it's free to use. It runs on Mac, Windows, and Ubuntu Linux, and it's licensed under the BSD license.



Features

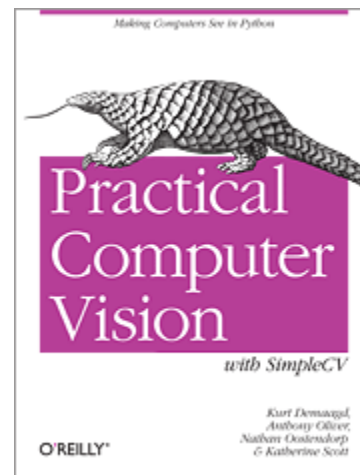
4

- ❑ Convenient "Superpack" installation for rapid deployment
- ❑ Feature detection and discrimination of Corners, Edges, Blobs, and Barcodes
- ❑ Filter and sort image features by their location, color, quality, and/or size
- ❑ An integrated iPython interactive shell makes developing code easy
- ❑ Image manipulation and format conversion
- ❑ Capture and process video streams from Kinect, Webcams, Firewire, IP Cams, or even mobile phones

New (and only) book!

5

- ❑ Learn how to build your own computer vision (CV) applications quickly and easily with SimpleCV.
- ❑ You can access the book online through the Safari collection of the ITESM CEM Digital Library.



6

SimpleCV Framework

Installation (Ubuntu Linux)

7

- Installing SimpleCV for Ubuntu is done through a .deb package. From the SimpleCV download page, click the latest stable release link. This will download the package and handle the installation of all the required dependencies.
- From the command line, type the following two commands:

```
$ sudo apt-get install ipython python-opencv python-scipy python-numpy python-pygame python-setuptools python-pip
$ sudo pip install
https://github.com/ingenuitas/SimpleCV/zipball/master
```
- The first command installs the required Python packages, and the second command uses the newly installed pip tool to install the latest version of SimpleCV from the GitHub repository.
- Once everything is installed, you type 'simplecv' on the command line to launch the SimpleCV interactive shell.

Installation (Ubuntu Linux)

8

- Note, however, that even recent distributions of Ubuntu may have an outdated version of OpenCV, one of the major dependencies for SimpleCV. If the installation throws errors with OpenCV, in a Terminal window enter:

```
$ sudo add-apt-repository ppa:gijzelaar/opencv2.3  
$ sudo apt-get update
```

- Once SimpleCV is installed, start the SimpleCV interactive Python shell by opening a command prompt and entering `python -m SimpleCV.__init__`. A majority of the examples can be completed from the SimpleCV shell.

Hello World

9

```
from SimpleCV import Camera, Display, Image ①
# Initialize the camera
cam = Camera()
# Initialize the display
display = Display() ②
# Snap a picture using the camera
img = cam.getImage() ③
# Show the picture on the screen
img.save(display) ④
```

Retrieve an Image-object from the camera with the highest quality possible.

Hello world (2)

10

```
from SimpleCV import Camera, Display, Image
import time
# Initialize the camera
cam = Camera()
# Initialize the display
display = Display()
# Snap a picture using the camera
img = cam.getImage()
# Show some text
img.drawText("Hello World!")
# Show the picture on the screen
img.save(display)
# Wait five seconds so the window doesn't close
right away
time.sleep(5)
```

The SimpleCV Shell

- The shell is built using IPython, an interactive shell for Python development.
- Most of the example code in this book is written so that it could be executed as a standalone script. However, this code can be still be run in the SimpleCV shell.
- The interactive tutorial is started with the tutorial command:
- `>>> tutorial`

A screenshot of the SimpleCV shell

12

```
+-----+
SimpleCV 1.2.0 [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:1> █
```

Introduction to the camera

- The simplest setup is a computer with a built-in webcam or an external video camera. These usually fall into a category called a USB Video Class (UVC) device.
 - ▣ UVC has emerged as a “device class” which provides a standard way to control video streaming over USB.
 - ▣ Most webcams today are now supported by UVC, and do not require additional drivers for basic operation.
 - ▣ Not all UVC devices support all functions, so when in doubt, tinker with the camera in a program like `guvcview` to see what works and what does not.

RPi Verified Peripherals: USB Webcams

14

Brand	Name	Model Number	Hardware ID	Verified OS	Verified OS version	Verified Resolution	Additional Information
Logitech	Webcam C100	V-U0013		raspbian/wheezy	2012-08-16		works fine without powered hub
Logitech	Webcam C200		046d:0802				
Logitech	Webcam C210		046d:0819	Raspbian/wheezy	2012-12-16	320x240, 640x480	Works fine without external power.
Logitech	Webcam C270						With external power
Logitech	Webcam C310						Does not require a powered hub to capture snapshots
Logitech	Webcam C510						
Logitech	Webcam C525						Works fine without powered hub
Logitech	Webcam C615						Works fine without powered hub
Logitech	Webcam C905		046d:080a	occidentalis	v0.2	1600x1200	Works fine without powered hub, she is uncvideo and detected out of box as Video0 V4L device. 1600x1200 is slow rate but he tested with motion.
Logitech	Webcam C910						With external power, is uncvideo. 320x240 works powered directly by the Raspberry Pi.
Logitech	Webcam C920			raspbian/wheezy			With powered hub, detected out of box as Video0 V4L device
Logitech	QuickCam Orbit/Sphere						Works with external power
Logitech	QuickCam Pro 9000			raspbian/wheezy			Powered by RasPi
Logitech	QuickCam Pro for Notebooks	960-000047	046d:0991	Raspbian Wheezy	2012-12-16	160x120 320x240 640x480	With guvcview it shows at about 4fps at 160x120, and at about 1fps at 640x480. GUVViewer Controls are available for focus and exposure.
Logitech	QuickCam Ultra Vision			raspbian/wheezy			Powered by RasPi
Logitech	Webcam Pro 4000						It uses pwc driver which does not work. Maybe it's because of general Raspberry Pi USB bug.
Logitech	Webcam Pro 9000		046d:0809	Arch Linux			Powered by RasPi, measured ~120 mA current capturing at ~5fps. Has issues capturing images at higher than default resolutions (using motion - Arch and Debian).

Camera Initialization

15

- The following line of code initializes the camera:

```
from SimpleCV import Camera
# Initialize the camera
cam = Camera()
```

- This approach will work when dealing with just one camera, using the default camera resolution, without needing any special calibration.

Simplified camera initialization

16

- Shortcut when the goal is simply to initialize the camera and make sure that it is working:

```
from SimpleCV import Camera
# Initialize the camera
cam = Camera()
# Capture and image and display it
cam.getImage().show()
```

- The `show()` function simply pops up the image from the camera on the screen. It is often necessary to store the image in a variable for additional manipulation instead of simply calling `show()` after `getImage()`, but this is a good block of code for a quick test to make sure that the camera is properly initialized and capturing video.

Example output of the basic camera example

17



Accessing more than one camera

18

- To access more than one camera, pass the `camera_id` as an argument to the `Camera()` constructor.
- On Linux, all peripheral devices have a file created for them in the `/dev` directory. For cameras, the file names start with `video` and end with the camera ID, such as `/dev/video0` and `/dev/video1`.

```
from SimpleCV import Camera
# First attached camera
cam0 = Camera(0)
# Second attached camera
cam1 = Camera(1)
# Show a picture from the first camera
cam0.getImage().show()
# Show a picture from the second camera
cam1.getImage().show()
```

Forcing the resolution

19

- The SimpleCV framework can control many other camera properties. An easy example is forcing the resolution.
- Almost every webcam supports the standard resolutions of 320×240, and 640×480 (often called “VGA” resolution). Many newer webcams can handle higher resolutions such as 800×600, 1024×768, 1200×1024, or 1600×1200.
- Here is an example of how to move text to the upper left quadrant of an image, starting at the coordinates (160, 120) on a 640×480 image:

```
from SimpleCV import Camera
cam = Camera(0, { "width": 640, "height": 480 })
img = cam.getImage()
img.drawText("Hello World", 160, 120)
img.show()
```

- Notice that the camera’s constructor passed a new argument in the form of {"key":value}

20

Drawing on Images in SimpleCV

Layers

21

- Layers can be unlimited and what's ever above or on top of the other layer will cover the layer below it. Let's look at the simplecv logo with some text written on it.
- Which in reality is an image with 3 layers, each layer has the text displayed. If we rotate the image and expand the layers you can get a better idea of what is really happening with image layers.



Example

22

```
>>> scv = Image('simplecv')
>>> logo = Image('logo')
>>> #write image 2 on top of image
>>> scv.dl().blit(logo)
```

`dl(index=-1)`
SUMMARY

Alias for [getDrawingLayer\(\)](#)

`blit(img, coordinates=(0, 0))`

Blit one image onto the drawing layer at upper left coordinates

Parameters:

`img` - Image coordinates - Tuple



NOTE: Run **help DrawingLayer** for more information.

Text on the image

23

- Here is an example of how to move text to the upper left quadrant of an image, starting at the coordinates (160, 120) on a 640×480 image:

```
from SimpleCV import Camera
cam = Camera(0, { "width": 640, "height": 480 })
img = cam.getImage()
img.drawText("Hello World", 160, 120)
img.show()
```

Example of Hello World application with the “Hello World” text

24



Camera class attributes

25

- The Camera() function has a properties argument for basic camera calibration.
- Multiple properties are passed in as a comma delimited list, with the entire list enclosed in brackets.
- Note that the camera ID number is NOT passed inside the brackets, since it is a separate argument. The configuration options are:
 - width and height
 - brightness
 - contrast
 - saturation
 - hue
 - gain
 - exposure
- The available options are part of the computer's UVC system.

A live camera feed

26

- To get live video feed from the camera, use the `live()` function.

```
from SimpleCV import Camera
cam = Camera()
cam.live()
```

- In addition to displaying live video feed, the `live()` function has two other very useful properties. The live feed makes it easy to find both the coordinates and the color of a pixel on the screen.
 - ▣ To get the coordinates or color for a pixel, use the `live()` function as outlined in the example above.
 - ▣ After the window showing the video feed appears, click the left mouse button on the image for the pixel of interest.
 - ▣ The coordinates and color for the pixel at that location will then be displayed on the screen and also output to the shell. The coordinates will be in the `(x, y)` format, and the color will be displayed as an RGB triplet `(R,G,B)`.

Demonstration of the live feed

27



Display object's isDone() function

28

- To control the closing of a window based on the user interaction with the window:

```
from SimpleCV import Display, Image
import time
display = Display()
Image("logo").save(display)
print "I launched a window"
# This while loop will keep looping until
the window is closed
while not display.isDone():
    time.sleep(0.1)
print "You closed the window"
```

The user will not be able to close the window by clicking the close button in the corner of the window.

Checks the event queue and returns True if a quit event has been issued.

It outputs to the command prompt, and not the image.

Information about the mouse

29

While the window is open, the following information about the mouse is available:

- `mouseX` and `mouseY` (Display class)
 - ▣ The coordinates of the mouse
- `mouseLeft`, `mouseRight`, and `mouseMiddle`
 - ▣ Events triggered when the left, right, or middle buttons on the mouse are clicked
- `mouseWheelUp` and `mouseWheelDown`
 - ▣ Events triggered then the scroll wheel on the mouse is moved

How to draw on a screen

30

```
from SimpleCV import Display, Image, Color
winsize = (640,480)
display = Display(winsize)
img = Image(winsize)
img.save(display)

while not display.isDone():
    if display.mouseLeft:
        img.dl().circle((display.mouseX,display.mouseY),4,
            Color.WHITE, filled=True)
        img.save(display)
        img.save("painting.png")
```

If the button is clicked, draw the circle. The image has a drawing layer, which is accessed with the dl() function. The drawing layer then provides access to the circle() function.

Example using the drawing application

31

- The little circles from the drawing act like a paint brush, coloring in a small region of the screen wherever the mouse is clicked.



32

Examples

Time-Lapse Photography

33

```
from SimpleCV import Camera, Image
import time
cam = Camera()
# Set the number of frames to capture
numFrames = 10
# Loop until we reach the limit set in numFrames
for x in range(0, numFrames):
    img = cam.getImage()
    filepath = "image-" + str(x) + ".jpg"
    img.save(filepath)
    print "Saved image to: " + filepath
time.sleep(60)
```

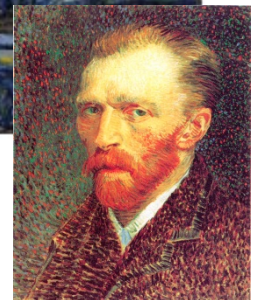
34

Color

Introduction

35

- Although color sounds like a relatively straightforward concept, different representations of color are useful in different contexts.
- The following examples work with an image of *The Starry Night* by Vincent van Gogh (1889).



getPixel

36

- In the SimpleCV framework, the colors of an individual pixel are extracted with the `getPixel()` function.

```
from SimpleCV import Image
img = Image('starry_night.png')
print img.getPixel(0, 0)
```

Prints the RGB triplet for the pixel at (0,0), which will equal (71.0, 65.0, 54.0).

Example RGB

37



Original Image



R-Component



G-Component



B-Component

HSV

38

- One criticism of RGB is that it does not specifically model luminance.
 - ▣ Yet the luminance/brightness is one of the most common properties to manipulate.
 - ▣ In theory, the luminance is the relationship of the of R, G, and B values.
 - ▣ In practice, however, it is sometimes more convenient to separate the color values from the luminance values.
- The solution is HSV, which stands for hue, saturation, and value.
 - ▣ The color is defined according to the hue and saturation, while value is the measure of the luminance/brightness.
 - ▣ The HSV color space is essentially just a transformation of the RGB color space because all colors in the RGB space have a corresponding unique color in the HSV space, and vice versa.

Example HSV

39



Original Image



Hue



Saturation

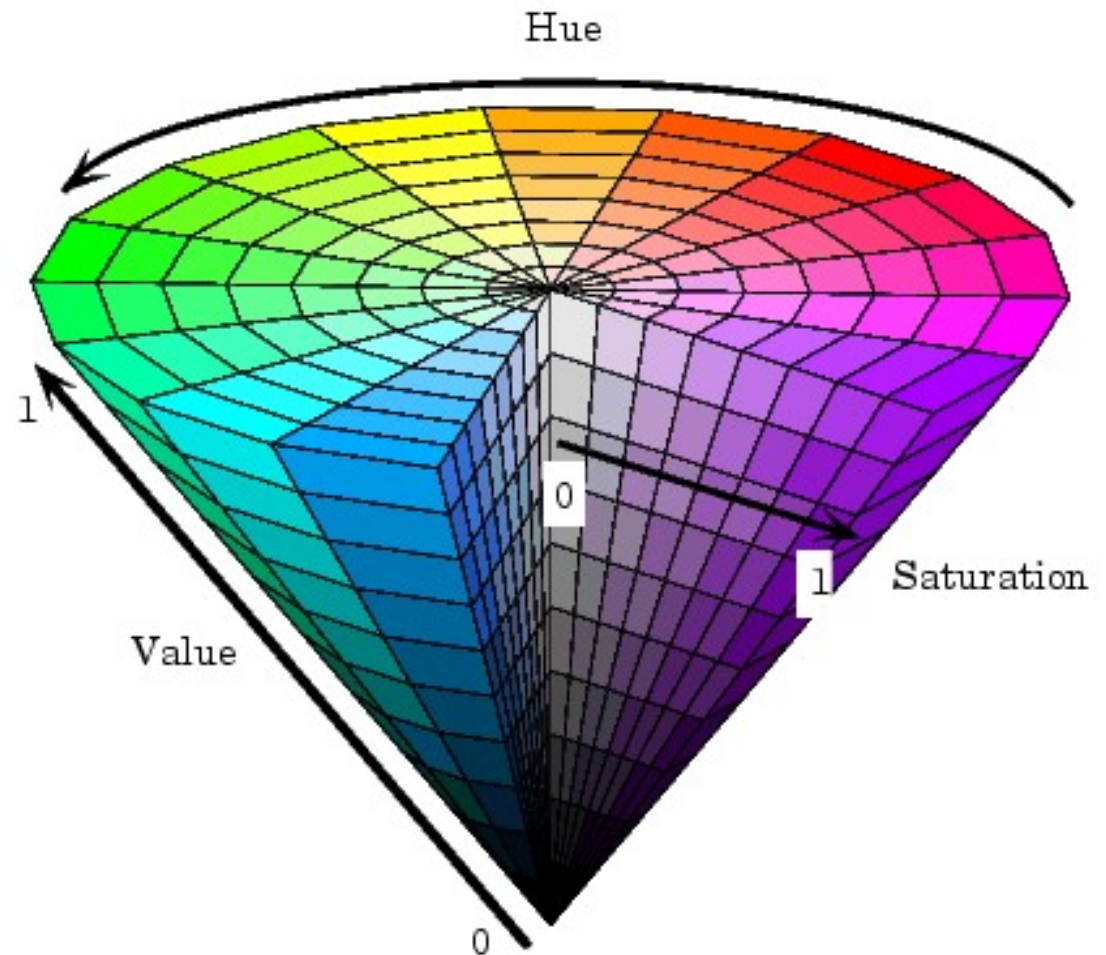


Value (Intensity)

RGB \Leftrightarrow HSV

40

The HSV color space is often used by people because it corresponds better to how people experience color than the RGB color space does.



RGB \Leftrightarrow HSV (2)

41

- It is easy to convert images between the RGB and HSV color spaces, as is demonstrated below.

```
from SimpleCV import Image
img = Image('starry_night.png')
hsv = img.toHSV()
print hsv.getPixel(25,25)
rgb = hsv.toRGB()
print rgb.getPixel(25,25)
```

It converts the image from the original RGB to HSV.

it prints the HSV values for the pixel

It converts the image back to RGB

it prints the HSV values for the pixel

RGB \Leftrightarrow HSV (2)

42

- The HSV color space is particularly useful when dealing with an object that has a lot of specular highlights or reflections.
 - ▣ In the HSV color space, specular reflections will have a high luminance value (V) and a lower saturation (S) component.
 - ▣ The hue (H) component may get noisy depending on how bright the reflection is, but an object of solid color will have largely the same hue even under variable lighting.

Grayscale

43

- A grayscale image represents the luminance of the image, but lacks any color components.
- An 8-bit grayscale image has many shades of grey, usually on a scale from 0 to 255.
 - ▣ The challenge is to create a single value from 0 to 255 out of the three values of red, green, and blue found in an RGB image.
 - There is no single scheme for doing this, but it is done by taking a weighted average of the three.

```
from SimpleCV import Image
img = Image('starry_night.png')
gray = img.grayscale()
print gray.getPixel(0,0)
```

Grayscale (2)

44

- `getPixel` returns the same number three times.
 - ▣ This keeps a consistent format with RGB and HSV, which both return three values.
- To get the grayscale value for a particular pixel without having to convert the image to grayscale, use `getGrayPixel()`.



The Starry Night, converted to grayscale

Color and Segmentation

- Segmentation is the process of dividing an image into areas of related content.
- Color segmentation is based on subtracting away the pixels that are far away from the target color, while preserving the pixels that are similar to the color.
- The Image class has a function called `colorDistance()` that computes the distance between every pixel in an image and a given color.
 - ▣ This function takes as an argument the RGB value of the target color, and it returns another image representing the distance from the specified color.

Segmentation Example

46

```
from SimpleCV import Image, Color
yellowTool = Image("yellowtool.png")
yellowDist = yellowTool.colorDistance((223, 191, 29))1
yellowDistBin = yellowDist.binarize(50).invert()2
onlyYellow = yellowTool - yellowDistBin3
onlyYellow.show()
```



1



2



3

47

Basic Feature Detection

Introduction

- The human brain does a lot of pattern recognition to make sense of raw visual inputs.
 - ▣ After the eye focuses on an object, the brain identifies the characteristics of the object —such as its shape, color, or texture— and then compares these to the characteristics of familiar objects to match and recognize the object.
- In computer vision, that process of deciding what to focus on is called feature detection.
 - ▣ A feature can be formally defined as “one or more measurements of some quantifiable property of an object, computed so that it quantifies some significant characteristics of the object” (Kenneth R. Castleman, *Digital Image Processing*, Prentice Hall, 1996).
 - ▣ Easier way to think of it:
 - a feature is an “interesting” part of an image.

Good vision system characteristics

49

- A good vision system should not waste time—or processing power—analyzing the unimportant or uninteresting parts of an image, so feature detection helps determine which pixels to focus on. In this session we will focus on the most basic types of features: blobs, lines, circles, and corners.
- If the detection is robust, a feature is something that could be reliably detected across multiple images.

Detection criteria

50

- How we describe the feature can also determine the situations in which we can detect the feature.
- Our detection criteria for the feature determines whether we can:
 - ▣ Find the features in different locations of the picture (position invariant)
 - ▣ Find the feature if it's large or small, near or far (scale invariant)
 - ▣ Find the feature if it's rotated at different orientations (rotation invariant)

Blobs

51

- Blobs are objects or connected components, regions of similar pixels in an image.
 - ▣ Examples:
 - ▣ a group of brownish pixels together, which might represent food in a pet food detector.
 - ▣ a group of shiny metal looking pixels, which on a door detector would represent the door knob
 - ▣ a group of matte white pixels, which on a medicine bottle detector could represent the cap.
- Blobs are valuable in machine vision because many things can be described as an area of a certain color or shade in contrast to a background.



Finding Blobs

52

- `findBlobs()` can be used to find objects that are **lightly** colored in an image. If no parameters are specified, the function tries to automatically detect what is bright and what is dark.



Left: Original image of pennies; Right: Blobs detected

Blob measurements

53

After a blob is identified we can:

- measure a lot of different things:
 - ▣ area
 - ▣ width and height
- find the centroid
- count the number of blobs
- look at the color of blobs
- look at its angle to see its rotation
- find how close it is to a circle, square, or rectangle —or compare its shape to another blob

Blob detection and measurement

54

```
from SimpleCV import Image
pennies = Image("pennies.png")
binPen = pennies.binarize() ①
blobs = binPen.findBlobs() ②
blobs.show(width=5) ③
```

1. Blobs are most easily detected on a binarized image.
2. Since no arguments are being passed to the `findBlobs()` function, it returns a `FeatureSet`
 - ❑ list of features about the blobs found
 - ❑ has a set of defined methods that are useful when handling features
3. `show()` function is being called on blobs and not the `Image` object
 - ❑ It draws each feature in the `FeatureSet` on top of the original image and then displays the results.

Blob detection and measurement (3)

55

- After the blob is found, several other functions provide basic information about the feature, such as its size, location, and orientation.

```
from SimpleCV import Image
pennies = Image("pennies.png")
binPen = pennies.binarize()
blobs = binPen.findBlobs()
print "Areas: ", blobs.area()
print "Angles: ", blobs.angle()
print "Centers: ", blobs.coordinates()
```

Blob detection and measurement (2)

56

- ▣ area function: returns an array of the area of each feature in pixels.
 - By default, the blobs are sorted by size, so the areas should be ascending in size.
- ▣ angle function: returns an array of the angles, as measured in degrees, for each feature.
 - The angle is the measure of rotation of the feature away from the x-axis, which is the 0 point. (+: counter-clockwise rotation; - : , clockwise rotation)
- ▣ coordinates function: returns a two-dimensional array of the (x, y) coordinates for the center of each feature.

Finding Dark Blobs

57

- If the objects of interest are darkly colored on a light background, use the `invert()` function.

```
from SimpleCV import Image
img = Image("chessmen.png")
invImg = img.invert() ①
blobs = invImg.findBlobs() ②
blobs.show(width=2) ③
img.addDrawingLayer(invImg.dl()) ④
img.show()
```



Finding Dark Blobs (2)

58

1. `invert()` function: turns the black chess pieces white and turns the white background to black
2. `findBlobs()` function: can then find the lightly colored blobs as it normally does.
3. `Show` the blobs. Note, however, that this function will show the blobs on the inverted image, not the original image.
4. To make the blobs appear on the original image, take the drawing layer from the inverted image (which is where the blob lines were drawn), and add that layer to the original image.

Finding Blobs of a Specific Color

59

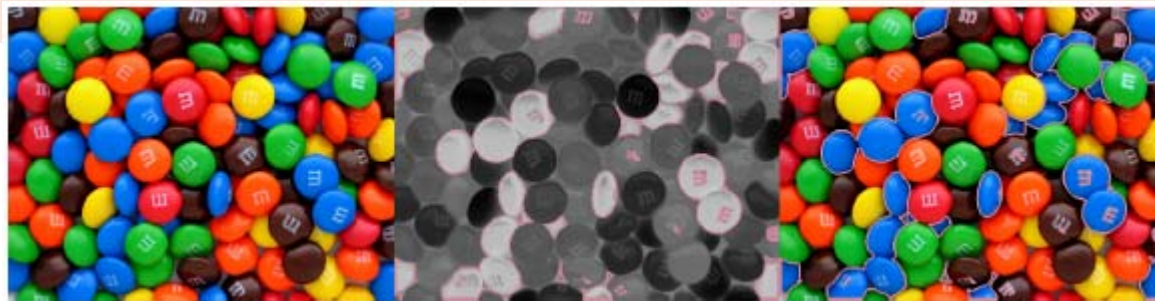
- In many cases, the actual color is more important than the brightness or darkness of the objects.
- Example: find the blobs that represent the blue candies



Finding Blobs of a Specific Color (2)

60

```
from SimpleCV import Color, Image
img = Image("mandms.png")
blue_distance =
img.colorDistance(Color.BLUE).invert() ①
blobs = blue_distance.findBlobs() ②
blobs.draw(color=Color.PUCE, width=3) ③
blue_distance.show()
img.addDrawingLayer(blue_distance.dl()) ④
img.show()
```



Left: the original image; Center: blobs based on the blue distance; Right: The blobs on the original image

Finding Blobs of a Specific Color (3)

61

1. `colorDistance()` function: returns an image that shows how far away the colors in the original image are from the passed in `Color.BLUE` argument.
 - ❑ To make this even more accurate, we could find the RGB triplet for the actual blue color on the candy.
 - ❑ Because any colors close to blue are black and colors far away from blue are white, we again use the `invert()` function to switch the target blue colors to white instead.
2. We use the new image to find the blobs representing the blue candies.
 - ❑ We can also fine-tune what the `findBlobs()` function discovers by passing in a threshold argument. The threshold can either be an integer or an RGB triplet. When a threshold value is passed in, the function changes any pixels that are darker than the threshold value to white and any pixels above the value to black.

Finding Blobs of a Specific Color (4)

3. In the previous examples, we have used the `FeatureSet show()` method instead of these two lines (`blobs.show()`). That would also work here. We've broken this out into the two lines here just to show that they are the equivalent of using the other method.
 - ❑ To outline the blue candies in a color not otherwise found in `candy`, they are drawn in puce, which is a reddish color.
4. Similar to the previous example, the drawing ends up on the `blue_distance` image. Copy the drawing layer back to the original image.

Blob detection in less-than-ideal light conditions

63

- ❑ Sometimes the lighting conditions can make color detection more difficult.
- ❑ To resolve this problem use `hueDistance()` instead of `colorDistance()`.
 - ❑ The hue is more robust to changes in light

```
from SimpleCV import Color, Image
img = Image("mandms-dark.png")
blue_distance = img.hueDistance(Color.BLUE).invert() 1
blobs = blue_distance.findBlobs()
blobs.draw(color=Color.PUCE, width=3)
img.addDrawingLayer(blue_distance.dl())
img.show()
```

Blob detection in less-than-ideal light conditions (2)

64



*Blobs detected with
hueDistance()*

*Blobs detected with
colorDistance()*

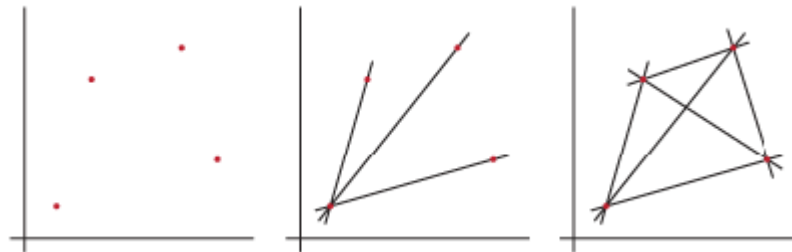
65

Lines and Circles

Lines

66

- A line feature is a straight edge in an image that usually denotes the boundary of an object.
- The calculations involved for identifying lines can be a bit complex. The reason is
 - ▣ because an edge is really a list of (x, y) coordinates, and any two coordinates could possibly be connected by a straight line.



Left: Four coordinates; Center: One possible scenario for lines connecting the points; Right: An alternative scenario

Hough transform



- The way this problem is handled behind-the-scenes is by using the Hough transform technique.
- This technique effectively looks at all of the possible lines for the points and then figures out which lines show up the most often. The more frequent a line is, the more likely the line is an actual feature.

findLines() function

68

Utilizes the Hough transform and returns a FeatureSet of the lines found

- `coordinates()`

- ▣ Returns the (x, y) coordinates of the starting point of the line(s).

- `width()`

- ▣ Returns the width of the line, which in this context is the difference between the starting and ending x coordinates of the line.

- `height()`

- ▣ Returns the height of the line, or the difference between the starting and ending y coordinates of the line.

- `length()`

- ▣ Returns the length of the line in pixels.

findLines() Example

69

- The example looks for lines on a block of wood.

```
from SimpleCV import Image
img = Image("block.png")
lines = img.findLines()
lines.draw(width=3)
img.show()
```

- The findLines() function returns a FeatureSet of the line features.
- This draws the lines in green on the image, with each line having a width of 3 pixels.



findLines() tuning parameters

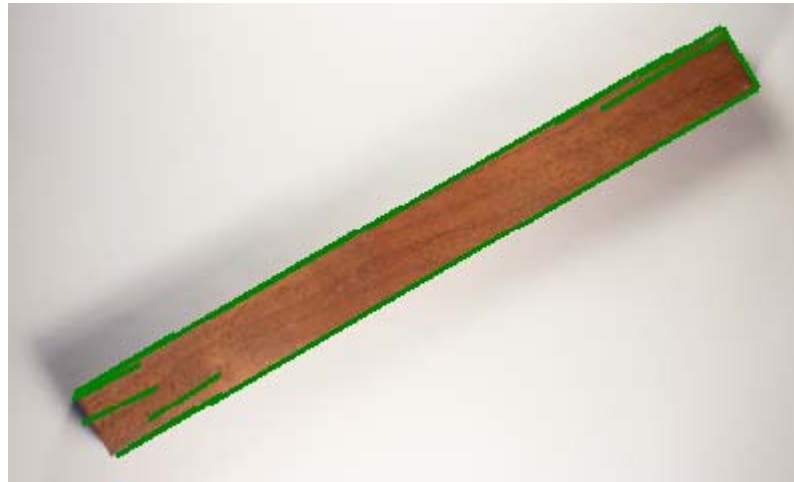
70

- Threshold
 - ▣ This sets how strong the edge should be before it is recognized as a line (default = 80)
- Minlinelength
 - ▣ Sets what the minimum length of recognized lines should be.
- Maxlinegap
 - ▣ Determines how much of a gap will be tolerated in a line.
- Cannyth1
 - ▣ This is a threshold parameter that is used with the edge detection step. It sets what the minimum “edge strength” should be.
- Cannyth2
 - ▣ This is a second parameter for the edge detection which sets the “edge persistence.”

findLines() Example new threshold

71

```
from SimpleCV import Image
img = Image("block.png")
# Set a low threshold
lines = img.findLines(threshold=10)
lines.draw(width=3)
img.show()
```



Line detection at a lower threshold

Circles

72

- The method to find circular features is called `findCircle()`
- It returns a `FeatureSet` of the circular features it finds, and it also has parameters to help set its sensitivity.

findCircle() parameters

73

- Canny
 - ▣ This is a threshold parameter for the Canny edge detector. The default value is 100. If this is set to a lower number, it will find a greater number of circles. Higher values instead result in fewer circles.
- Thresh
 - ▣ This is the equivalent of the threshold parameter for findLines(). It sets how strong an edge must be before a circle is recognized. The default value for this parameter is 350.
- Distance
 - ▣ Similar to the maxlinegap parameter for findLines(). It determines how close circles can be before they are treated as the same circle. If left undefined, the system tries to find the best value, based on the image being analyzed.

findCircle() FeatureSet

74

- radius()
- diameter()
- perimeter()
 - ▣ It may seem strange that this isn't called circumference, but the term perimeter makes more sense when dealing with a non-circular features. Using perimeter here allows for a standardized naming convention.

findCircle() Example

75

```
• from SimpleCV import Image
• img = Image("pong.png")
1. circles =
    img.findCircle(canny=200,thresh=250,distance=15)
2. circles = circles.sortArea()
3. circles.draw(width=4)
4. circles[0].draw(color=Color.RED, width=4)
5. img_with_circles = img.applyLayers()
6. edges_in_image = img.edges(t2=200)
7. final =
    img.sideBySide(edges_in_image.sideBySide(img_with
    _circles)).scale(0.5)
• final.show()
```

applyLayers() Render all of the layers onto the current image and return the result. Indices can be a list of integers specifying the layers to be used.
sideBySide() Combine two images as a side by side images.

findCircle() Example (2)

76



Image showing the detected circles

Corners

77

- are places in an image where two lines meet.
- unlike edges, are relatively unique and effective for identifying parts of an image.
 - ▣ For instance, when trying to analyze a square, a vertical line could represent either the left or right side of the square.
 - ▣ Likewise, detecting a horizontal line can indicate either the top or the bottom.
 - ▣ Each corner is unique. For example, the upper left corner could not be mistaken for the lower right, and vice versa. This makes corners helpful when trying to uniquely identify certain parts of a feature.
- Note: a corner does not need to be a right angle at 90 degrees

findCorners() function

78

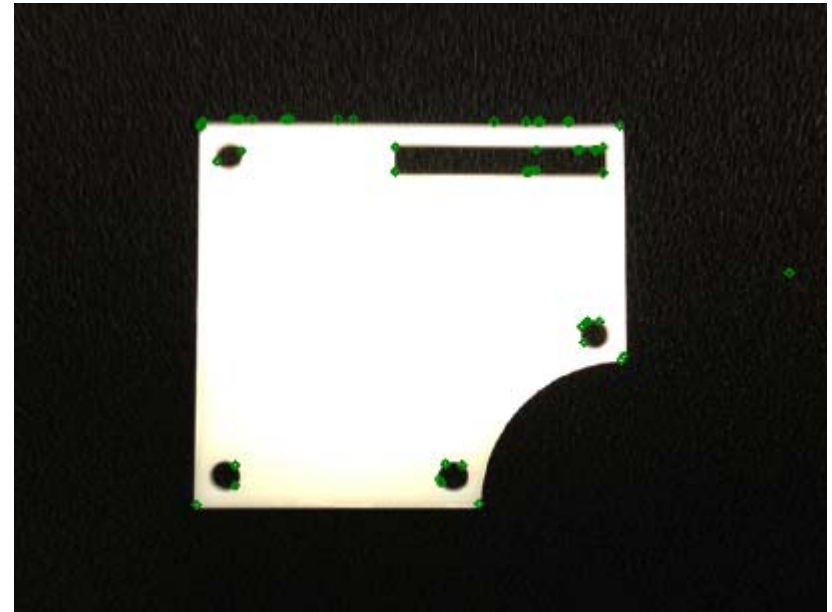
- analyzes an image and returns the locations of all of the corners it can find
- returns a FeatureSet of all of the corner features it finds
- has parameters to help fine-tune the corners that are found in an image.

findCorners() Example

79

```
from SimpleCV import Image
= Image( 'corners.png' )
imgimg.findCorners.show( )
```

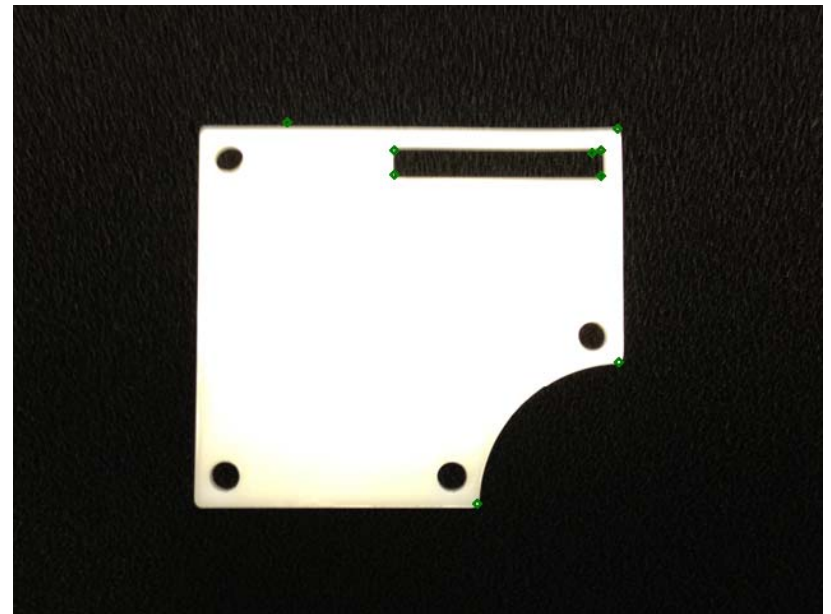
- Notice that the example finds a lot of corners (default: 50)
- Based on visual inspection, it appears that there are four main corners.
- To restrict the number of corners returned, we can use the `maxnum` parameter.



findCorners() Example (2)

80

```
from SimpleCV import Image
img = Image('corners.png')
img.findCorners.(maxnum=9).show()
```



*Limiting findCorners() to a maximum of
nine corners*

81

The XBox Kinect

Introduction

- Historically, the computer vision market has been dominated by 2D vision systems.
- 3D cameras were often expensive, relegating them to niche market applications.
- More recently, however, basic 3D cameras have become available on the consumer market, most notably with the XBox Kinect.
 - ▣ The Kinect is built with two different cameras.
 - ▣ The first camera acts like a traditional 2D 640×480 webcam.
 - ▣ The second camera generates a 640×480 depth map, which maps the distance between the camera and the object.
 - ▣ This obviously will not provide a Hollywood style 3D movie, but it does provide an additional degree of information that is useful for things like feature detection, 3D modeling, and so on.

Installation

- The Open Kinect project provides free drivers that are required to use the Kinect.
- The standard installation on both Mac and Linux includes the Freenect drivers, so no additional installation should be required.
- For Windows users, however, additional drivers must be installed.
- Because the installation requirements from Open Kinect may change, please see their website for installation requirements at <http://openkinect.org>.

Using the Kinect

84

- The overall structure of working with the 2D camera is similar to a local camera. However, initializing the camera is slightly different:

```
from SimpleCV import Kinect
# Initialize the Kinect
kin = Kinect()
# Snap a picture with the Kinect
img = kin.getImage()
img.show()
```

Kinect() constructor
(does not take any
arguments)

snap a picture
with the Kinect's
2D camera

Depth Information Extraction (1)

85

- Using the Kinect simply as a standard 2D camera is a pretty big waste of money. The Kinect is a great tool for capturing basic depth information about an object.
 - ▣ It measures depth as a number between 0 and 1023, with 0 being the closest to the camera and 1023 being the farthest away.
 - ▣ SimpleCV automatically scales that range down to a 0 to 255 range.
 - Why? Instead of treating the depth map as an array of numbers, it is often desirable to display it as a grayscale image. In this visualization, nearby objects will appear as dark grays, whereas objects in the distance will be light gray or white.

Depth Information Extraction (2)

86

```
from SimpleCV import Kinect
# Initialize the Kinect
kin = Kinect()
# This works like getImage, but returns depth information
depth = kin.getDepth()
depth.show()
```

- The Kinect's depth map is scaled so that it can fit into a 0 to 255 grayscale image.
- This reduces the granularity of the depth map.



A depth image from the Kinect

Getting the original range depth

87

- It is possible to get the original 0 to 1023 range depth map.
- The function `getDepthMatrix()` returns a NumPy matrix with the original full range of depth values.
 - ▣ This matrix represents the 2×2 grid of each pixel's depth.

```
from SimpleCV import Kinect
# Initialize the Kinect
kin = Kinect()
# This returns the 0 to 1023 range depth map
depthMatrix = kin.getDepthMatrix()
print depthMatrix
```

Kinect Example: real-time depth camera video feed

88

```
from SimpleCV import Kinect
# Initialize the Kinect
kin = Kinect()
# Initialize the display
display = kin.getDepth().show()
# Run in a continuous loop forever
while (True):
    # Snaps a picture, and returns the grayscale depth map
    depth = kin.getDepth()
    # Show the actual image on the screen
    depth.save(display)
```


89

Networked Cameras

Introduction

- The previous examples in this lecture have assumed that the camera is directly connected to the computer.
- However, SimpleCV can also control Internet Protocol (IP) Cameras.
 - ▣ Popular for security applications, IP cameras contain a small **web server** and a camera sensor.
 - They stream the images from the camera over a web feed.
 - ▣ These cameras have recently dropped substantially in price.
 - Low end cameras can be purchased for as little as \$30 for a wired camera and \$60 for a wireless camera.

IP Camera Advantages

- ❑ Two-way audio via a single network cable allows users to communicate with what they are seeing.
- ❑ Flexibility: IP cameras can be moved around anywhere on an IP network (including wireless).
- ❑ Distributed intelligence: with IP cameras, **video analytics** can be placed in the camera itself allowing scalability in analytics solutions.
- ❑ Transmission of commands for PTZ (pan, tilt, zoom) cameras via a single network cable.

IP Camera Advantages (2)

- ❑ Encryption & authentication: IP cameras offer secure data transmission through encryption and authentication methods such as WEP, WPA, WPA2, TKIP, AES.
- ❑ Remote accessibility: live video from selected cameras can be viewed from any computer, anywhere, and also from many mobile smartphones and other devices.
- ❑ IP cameras are able to function on a wireless network.
- ❑ PoE - Power over ethernet. Modern IP cameras have the ability to operate without an additional power supply. They can work with the PoE-protocol which gives power via the ethernet-cable

IP Camera potential disadvantages

93

- Higher initial cost per camera, except where cheaper webcams are used.
- High network bandwidth requirements: a typical CCTV camera with resolution of 640x480 pixels and 10 frames per second (10 frame/s) in MJPEG mode requires about 3 Mbit/s.
- As with a CCTV/DVR system, if the video is transmitted over the public Internet rather than a private IP LAN, the system becomes open to a wider audience of hackers and hoaxers.
 - ▣ Criminals can hack into a CCTV system to observe security measures and personnel, thereby facilitating criminal acts and rendering the surveillance counterproductive.

Accessing a MJPG stream (1)

94

- Most IP cameras support a standard HTTP transport mode, and stream video via the Motion JPEG (MJPG) format.
- To access a MJPG stream, use the `JpegStreamCamera` library.
- The basic setup is the same as before, except that now the constructor must provide
 - the **address of the camera** and
 - the **name of the MJPG file**.

MJPG: video format in which each video frame or interlaced field of a digital video sequence is separately compressed as a JPEG image.

Accessing a MJPG stream (2)

95

- In general, initializing an IP camera requires the following information:
 - ▣ The IP address or hostname of the camera (`mycamera`)
 - ▣ The path to the Motion JPEG feed (`video.mjpg`)
 - ▣ The username and password, if required.

```
from SimpleCV import JpegStreamCamera
# Initialize the webcam by providing URL to the camera
cam = JpegStreamCamera("http://mycamera/video.mjpg")
cam.getImage().show()
```

Having difficulty accessing an IP camera?

96

- ❑ Try loading the URL in a web browser. It should show the video stream.
- ❑ If the video stream does not appear, it may be that the URL is incorrect or that there are other configuration issues.
- ❑ One possible issue is that the URL requires a login to access it.

Authentication information

97

- If the video stream requires a username and password to access it, then provide that authentication information in the URL.

```
from SimpleCV import JpegStreamCamera
# Initialize the camera with login info in the URL
cam = JpegStreamCamera("http://admin:1234@192.168.1.10/video.mjpg")
cam.getImage().show()
```

Use your mobile device as IP camera

98

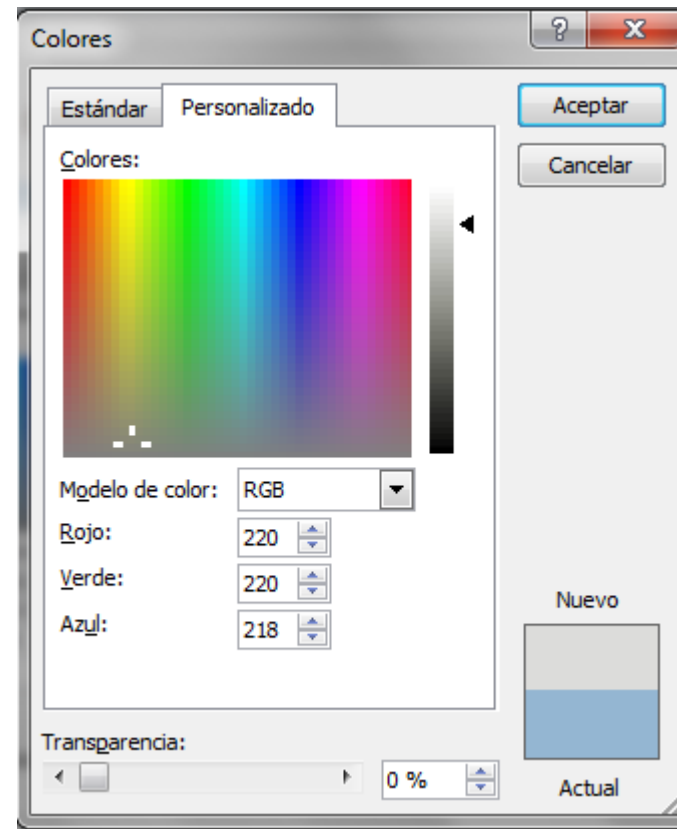
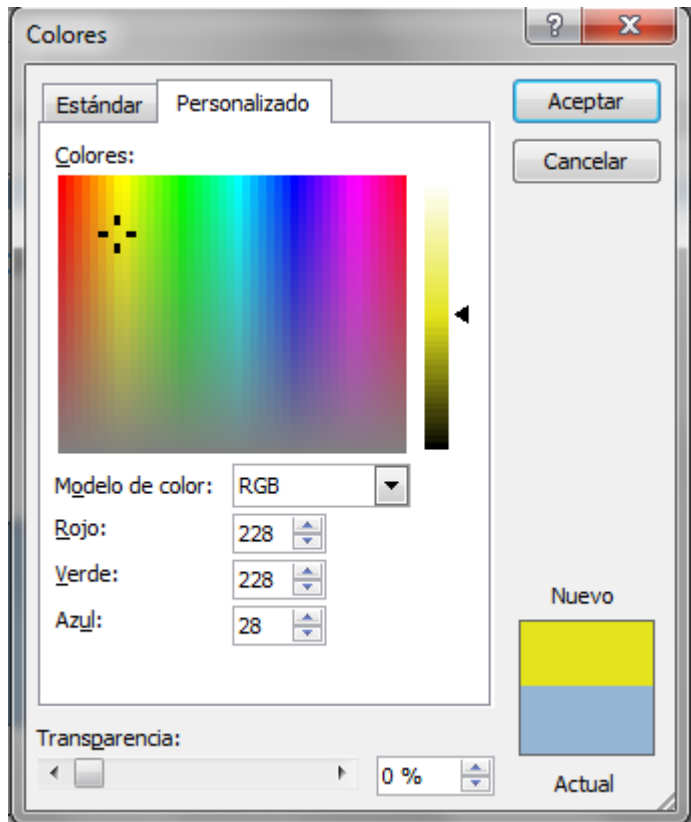
- Many phones and mobile devices today include a built-in camera.
- Tablet computers and both the iOS and Android smart phones can be used as network cameras with apps that stream the camera output to an MJPG server.
- To install one of these apps, search for “IP Cam” in the app marketplace on an iPhone/iPad or search for “IP Webcam” on Android devices.
- Some of these apps are for viewing feeds from other IP cameras, so **make sure that the app is designed as a server and not a viewer.**



IP Cam Pro
by Senstic



IP Webcam
by Pavel Khlebovich



100

Advanced Features

Introduction

101

- Previous lectures introduced feature detection and extraction, but mostly focused on common geometric shapes like lines and circles.
 - ▣ Even blobs assume that there is a contiguous grouping of similar pixels.
- This lecture builds on those results by looking for more complex objects. Most of this detection is performed by **searching for a template** of a known form in a larger image.
- This lecture also provides an overview of **optical flow**, which attempts to identify objects that change between two frames.

Topics

102

- Finding instances of template images in a larger image Using Haar classifiers, particularly to identify faces
- Barcode scanning for 1D and 2D barcodes
- Finding keypoints, which is a more robust form of template matching
- Tracking objects that move

Template Matching

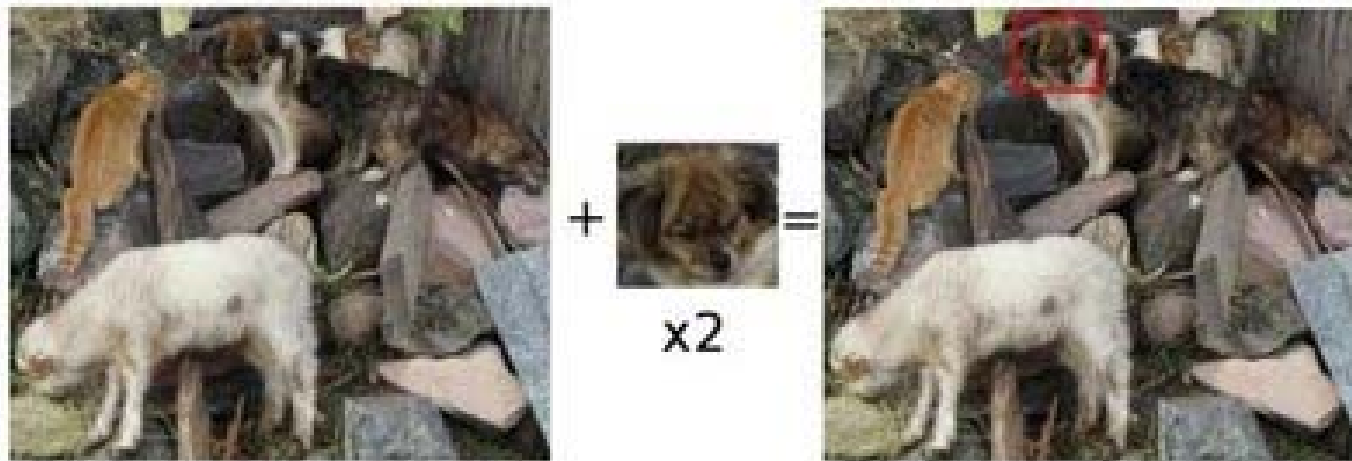
103

- Goal
 - ▣ learn how to use SimpleCV template matching functions to search for matches between an image patch and an input image
- What is template matching?
 - ▣ Template matching is a technique for finding areas of an image that match (are similar) to a template image (patch).

How does it work?

104

- We need two primary components:
 - ▣ Source image (I): The image in which we expect to find a match to the template image
 - ▣ Template image (T): The patch image which will be compared to the template image
- Our goal is to detect the highest matching area:



How does it work? (2)

105

- To identify the matching area, we have to compare the template image against the source image by sliding it:



- By sliding, we mean moving the patch one pixel at a time (left to right, up to down). At each location, a metric is calculated so it represents how “good” or “bad” the match at that location is (or how similar the patch is to that particular area of the source image).

Bitmap Template Matching

106

- This algorithm works by searching for instances where a bitmap template—a small image of the object to be found—can be found within a larger image.
 - ▣ For example, if trying to create a vision system to play Where's Waldo, the template image would be a picture of Waldo.
 - ▣ To match his most important feature, his face, the Waldo template would be cropped to right around his head and include a minimal amount of the background.
 - ▣ A template with his whole body would only match instances where his whole body was visible and positioned exactly the same way. The other component in template matching is the image to be searched.



findTemplate() function

107

- The template matching feature works by calling the `findTemplate()` function on the Image object to be searched.
- Next, pass as a parameter the template of the object to be found.



Pieces on a Go board

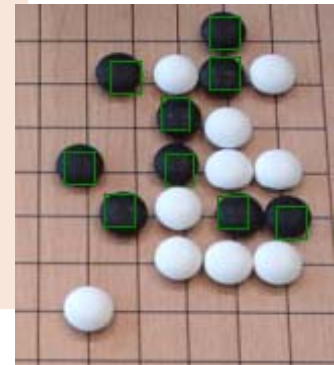


Template of black piece

findTemplate() function (2)

108

```
from SimpleCV import Image
# Get the template and image
goBoard = Image('go.png')
black = Image('go-black.png')
# Find the matches and draw them
matches = goBoard.findTemplate(black)
matches.draw()
# Show the board with matches print the number
goBoard.show()
print str(len(matches)) + " matches found."
# Should output: 9 matches found.
```



Matches for black pieces

findTemplate() function (3)

109

- The findTemplate() function also takes two optional arguments:
 - ▣ method
 - defines the algorithm to use for the matching
 - details about the various available matching algorithms can be found by typing `help Image.findTemplate` in the SimpleCV shell.
 - ▣ Threshold
 - fine-tunes the quality of the matches
 - works like thresholds with other feature matching functions
 - decreasing the threshold results in more matches, but could also result in more false positives.
- These tuning parameters can help the quality of results, but template matching is error prone in all but the most controlled, consistent environments.
- Keypoint matching, which is described in the next section, is a more robust approach.

Template Matching Methods

110

- Square difference matching method (CV_TM_SQDIFF)
 - These methods match the squared difference, so a perfect match will be 0 and bad matches will be large:

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

- Correlation matching methods(CV_TM_CCORR)
 - These methods multiplicatively match the template against the image, so a perfect match will be large and bad matches will be small or 0.

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

Template Matching Methods (2)

111

- Correlation coefficient matching methods (CV_TM_CCOEFF)
 - ▣ These methods match a template relative to its mean against the image relative to its mean, so a perfect match will be 1 and a perfect mismatch will be -1 ; a value of 0 simply means that there is no correlation (random alignments).

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I(x + x', y + y'))$$

where

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$$

$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$$

Normalized methods

112

- For each of the three methods just described, there are also normalized versions.
- The normalized methods are useful because they can help reduce the effects of lighting differences between the template and the image.
- In each case, the normalization coefficient is the same:

$$Z(x, y) = \sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x+x', y+y')^2}$$

Values of the method parameter for normalized template matching

113

Value of method parameter	Computed result
CV_TM_SQDIFF_NORMED	$R_{sq_diff_normed}(x, y) = \frac{R_{sq_diff}(x, y)}{Z(x, y)}$
CV_TM_CCORR_NORMED	$R_{ccor_normed}(x, y) = \frac{R_{ccor}(x, y)}{Z(x, y)}$
CV_TM_CCOEFF_NORMED	$R_{ccoeff_normed}(x, y) = \frac{R_{ccoeff}(x, y)}{Z(x, y)}$

- As usual, we obtain more accurate matches (at the cost of more computations) as we move from simpler measures (square difference) to the more sophisticated ones (correlation coefficient).

Limitations

- This approach supports searching for a wide variety of different objects but:
 - ▣ The image being searched always needs to be larger than the template image for there to be a match.
 - ▣ It is not scale or rotation invariant.
 - ▣ The size of the object in the template must equal the size it is in the image.
 - ▣ Lighting conditions also can have an impact because the image brightness or the reflection on specular objects can stop a match from being found.
- As a result, the matching works best when used in an extremely controlled environment.

Keypoint Template Matching

115

- Tracking an object as it moves between two images is a hard problem for a computer vision system.
 - ▣ This is a challenge that requires identifying an object in one image, finding that same object in the second image, and then computing the amount of movement.
- We can detect that something has changed in an image, using techniques such as subtracting one image from another.
- While the feature detectors we've looked at so far are useful for identifying objects in a single image, they are often sensitive to different degrees of rotation or variable lighting conditions.
- As these are both common issues when tracking an object in motion, a more robust feature extractor is needed. One solution is to use keypoints.

Keypoint Template Matching (2)

116

- A keypoint describes an object in terms that are independent of position, rotation, and lighting.
 - ▣ For example, in many environments, detecting corners make good keypoints.
 - As it was described before, a corner is made from two intersecting lines at any angle.
 - Move a corner from the top-left to the bottom-right of the screen and the angle between the two intersecting lines remains the same.
 - Rotate the image, and the angle remains the same.
 - Shine more light on the corner, and it's still the same angle.
 - Scale the image to twice its size, and again, the angle remains the same.
 - Corners are robust to many environmental conditions and image transformations.

findKeypoints()

117

- Keypoints can be more than just corners, but corners provide an intuitive framework for understanding the underpinnings of keypoint detection.
- The keypoints are extracted using the `findKeypoints()` function from the Image library.
- In addition to the typical feature properties, such as size and location, a keypoint also has a `descriptor()` function that outputs the actual points that are robust to location, rotation, scale, and so on.

findKeypoints() example

118

```
from SimpleCV import Image
card = Image('keycard.png')
keys = card.findKeypoints() ①
keys.draw() ②
card.show()
```

1. Find the keypoints on the image using the findKeypoints() method.
2. Draw those keypoints on the screen. They will appear as green circles at each point. The size of the circle represents the quality of the keypoint.



A common hotel room keycard



The extracted keypoints

findKeypoints() function example (2)

119

- The above keypoints are not particularly valuable on their own, but the next step is to apply this concept to perform template matching.
- As with the findTemplate() approach, a template image is used to find keypoint matches. However, unlike the previous example, keypoint matching does not work directly off the template image.
- Instead, the process extracts the keypoints from the template image, and then looks for those keypoints in the target image. This results in a much more robust matching system.

findKeypointMatch() limitations

120

- ❑ Keypoint matching works best when the object has a lot of texture with diverse colors and shapes.
- ❑ Objects with uniform color and simple shapes do not have enough keypoints to find good matches.
- ❑ The matching works best with small rotations, usually less than 45 degrees. Greater or larger rotations could work, but it will be harder for the algorithm to find a match.
- ❑ Whereas the findTemplate() function looks for multiple matches of the template, the [findKeypointMatch\(\)](#) function returns only the best match for the template.

findKeypointMatch() Arguments

121

- **template (required)**
 - The template image to search for in the target image.
- **quality**
 - Configures the quality threshold for matches.
 - Default: 500; range for best results: [300:500]
- **minDist**
 - The minimum distance between two feature vectors necessary in order to treat them as a match. The lower the value, the better the quality. Too low a value, though, prevents good matches from being returned.
 - Default: 0.2, range for best results: [0.05:0.3]
- **minMatch**
 - The minimum percentage of feature matches that must be found to match an object with the template.
 - Default: 0.4 (40%); good values typically range: [0.3:0.7]

findKeypointMatch() Arguments (2)

122

- Flavor: a string indicating the method to use to extract features.
 - “SURF” - extract the SURF features and descriptors. If you don’t know what to use, use this.
 - <http://en.wikipedia.org/wiki/SURF>
 - “STAR” - The STAR feature extraction algorithm
 - http://pr.willowgarage.com/wiki/Star_Detector
 - “FAST” - The FAST keypoint extraction algorithm
 - http://en.wikipedia.org/wiki/Corner_detection#AST_based_feature_detectors

drawKeypointMatches() function

123

- shows a side-by-side image, with lines drawn between the two images to indicate where it finds a match.
- If it draws lots of lines, the matching algorithm should work.



drawKeypointMatches() function (2)

125

- Note that keypoint matching only finds the single best match for the image. If multiple keycards were on the table, it would only report the closest match.



The detected match for the keycard

<http://www.youtube.com/watch?v=CxPeoQDc2-Y>

Optical Flow

- The next logical step beyond template matching is to understand how to track objects between frames.
- Optical flow is very similar to template matching in that it takes a small area of one image and then scans the same region in a second image in an attempt to find a match.
- If a match is found, the system indicates the direction of travel, as measured in (X, Y) points.
- Only the two-dimensional travel distance is recorded. If the object also moved closer to or further away from the camera, this distance is not recorded.

findMotion() function

127

- computes the optical flow
- has a single required parameter,
 - ▣ `previous_frame`, which is the image used for comparison.
- The previous frame must be the same size.

findMotion() function Example

128

```
from SimpleCV import Camera, Color, Display
cam = Camera()
previous = cam.getImage()
disp = Display(previous.size())
while not disp.isDone():
    current = cam.getImage()
    motion = current.findMotion(previous)
    for m in motion:
        m.draw(color=Color.RED,normalize=False)
    current.save(disp)
    previous = current
```

Draw the little red motion lines on the screen to indicate where the motion occurred.

Optical flow example

129



Optical flow example

- The findMotion function supports several different algorithms to detect motion. For additional information, type help Image in the SimpleCV shell.

<http://www.youtube.com/watch?v=V4r2HXGA8jw>

findMotion() parameters

130

- `previous_frame` - The last frame as an Image.
- `window` - The block size for the algorithm. For the the HS and LK methods this is the regular sample grid at which we return motion samples. For the block matching method this is the matching window size.
- `method` - The algorithm to use as a string. Your choices are:
 - 'BM' - default block matching robust but slow - if you are unsure use this.
 - http://en.wikipedia.org/wiki/Block-matching_algorithm
 - 'LK' - Lucas-Kanade method
 - http://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method
 - 'HS' - Horn-Schunck method
 - http://en.wikipedia.org/wiki/Horn%E2%80%93Schunck_method
- `aggregate` - If `aggregate` is true, each of our motion features is the average of motion around the sample grid defined by `window`. If `aggregate` is false we just return the value as sampled at the window grid interval. For block matching this flag is ignored.

Haar-like Features

131

- Unlike template matching, Haar-like Features are used to classify more generic objects.
- They are particularly popular for face detection, where the system determines whether an object is a generic face.
 - ▣ This is different from face recognition, which tries to identify whose face is in the image and is a much more complicated process.
 - ▣ Simply knowing that an object is a face is useful for segmenting the image, narrowing down a region of interest, or simply doing some other fun tricks.

Haar-like Features (2)

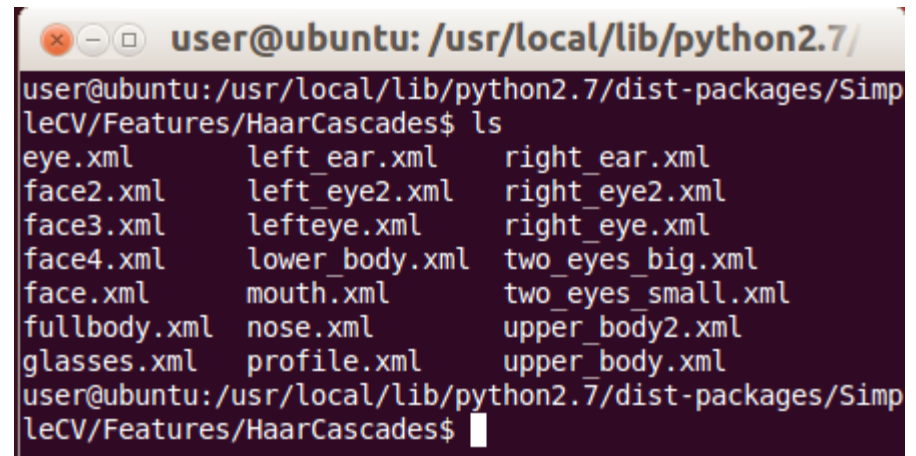
132

- Technically, Haar-like features refer to a way of slicing and dicing an image to identify the key patterns.
- The template information is stored in a file known as a Haar Cascade, usually formatted as an XML file.
 - ▣ This requires a fair amount of work to train a classifier system and generate the cascade file.
 - ▣ Fortunately, SimpleCV includes some common face detection cascades, and additional ones are available on the Internet.
- http://en.wikipedia.org/wiki/Haar-like_features
- <http://alereimondo.no-ip.org/OpenCV/34>

built-in cascades

133

- ❑ The built-in cascades with SimpleCV include:
 - ❑ Forward-looking faces
 - ❑ Profile faces
 - ❑ Eyes
 - ❑ Noses
 - ❑ Mouths
 - ❑ Ears
 - ❑ Upper and lower bodies



```
user@ubuntu: /usr/local/lib/python2.7/  
user@ubuntu: /usr/local/lib/python2.7/dist-packages/SimpleCV/Features/HaarCascades$ ls  
eye.xml          left_ear.xml     right_ear.xml  
face2.xml        left_eye2.xml    right_eye2.xml  
face3.xml        lefteye.xml      right_eye.xml  
face4.xml        lower_body.xml   two_eyes_big.xml  
face.xml         mouth.xml        two_eyes_small.xml  
fullbody.xml     nose.xml         upper_body2.xml  
glasses.xml     profile.xml      upper_body.xml  
user@ubuntu: /usr/local/lib/python2.7/dist-packages/SimpleCV/Features/HaarCascades$
```

findHaarFeatures() function

134

- Haar-like features are detected with the `findHaarFeatures()` function of the Image library.
- Parameters:
 - `cascade`
 - The path the Haar Cascade XML file.
 - `scale_factor`
 - The Haar Cascades are technically sensitive to the image's scale. To simulate scale invariance, the algorithm is run multiple times, scaling the template up with each run. The amount that the template should be scaled is controlled by the `scale_factor`. Default: 1.2 (it scales up the template 20% each time).
 - `min_neighbors`
 - This is like a threshold value in other feature detectors. Technically, it deals with the number of adjacent detections needed to classify the object, but conceptually it is easier to think of as a threshold parameter. Default: 2.
 - `use_canny`
 - The default value is to use Canny pruning to reduce false positives in potentially noisy images. This is usually always the best option.

findHaarFeatures() function example

135

```
from SimpleCV import Camera, Display
cam = Camera()
disp = Display(cam.getImage().size())
while disp.isNotDone():
    img = cam.getImage()
    # Look for a face
    faces = img.findHaarFeatures('face')
    if faces is not None:
        # Get the largest face
        faces = faces.sortArea()
        bigFace = faces[-1]
        # Draw a green box around the face
        bigFace.draw()
    img.save(disp)
```

loads one of the built-in Haar Cascade files



findHaarFeatures() function example (2)

136

- If working with a Haar Cascade that was downloaded from the Internet or created manually, simply specify the full path to the file. }
 - `findHaarFeatures("/path/to/haarcascade_frontalface_alt.xml")`
- The detection works like most other feature detectors. Simply pass the cascade file loaded earlier and it returns a FeatureSet with the matches.

<http://www.youtube.com/watch?v=c4LobbqeKZc>

<http://www.youtube.com/watch?v=aTErTqOIkss>

Limitations

137

- Haar Cascades also work best in controlled environments.
 - ▣ The Haar Cascade for detecting forward-looking faces will not work well on an image with a face looking to the side.
 - ▣ Size and rotation of the object can also create complications, though the underlying algorithms are more flexible than with template matching.

Barcode

138

- One-dimensional barcodes have become a ubiquitous part of the modern shopping experience.
- Two-dimensional barcodes are becoming increasingly popular for storing contact information, URL's, and other small bits of information.
- This growing popularity is helped by the availability of open source tools that can be used to extract information from these barcodes.
 - ▣ One example of these tools is the ZXing library (pronounced “Zebra Crossing”), which is an image processing library for barcodes.
 - ▣ The SimpleCV framework works with the ZXing library, making it easy to process images of one- and two-dimensional barcodes as part of a vision system.
 - ▣ As of SimpleCV version 1.3, ZXing is not automatically installed. If it is not installed, trying to use the barcode functions will throw a warning and not work. For instructions on installing ZXing, type `help Image.findBarcode` from the SimpleCV shell, or go to <http://code.google.com/p/zxing/>

findBarcode() function (1)

139

- ❑ To extract the information from a barcode, use the findBarcode() function.
- ❑ It works with both one-dimensional and two-dimensional barcodes.
- ❑ The image to be scanned can contain more than just a barcode, but the detector works best when there is not a lot of extra noise around it.
- ❑ In addition, while most barcodes are designed to work in spite of a partial obstruction of the barcode, the detector works best when it has a clear view of the barcode.

findBarcode() function (2)

140

```
from SimpleCV import Image, Barcode
# Load a one-dimensional barcode
img = Image('upc.png')
barcode = img.findBarcode()
print barcode.data
# Should output: 987654321098
```



Example of a one-dimensional barcode

```
from SimpleCV import Image, Barcode
# Load a QR barcode
img = Image('qr.png')
barcode = img.findBarcode()
print barcode.data.rstrip()
# Should output: http://www.simplecv.org
```

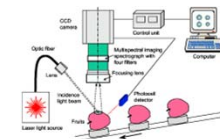
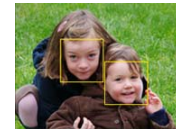


Example of a QR barcode

Why Learn Computer Vision?

141

- Computer vision is moving from a niche tool to an increasingly common tool for a diverse range of applications:
 - **facial recognition programs** or **gaming interfaces** like the Kinect
 - **automotive safety systems** where a car detects when the driver starts to drift from the lane, or when is getting drowsy
 - **point-and-shoot cameras** to help detect faces or other central objects to focus on
 - **high tech special effects** or **basic effects**, such as the virtual yellow first-and-ten line in football games, or the motion blurs on a hockey puck.
 - **industrial automation, biometrics, medicine,** and even **planetary exploration**
 - **food and agriculture**, where it is used to inspect and grade fruits and vegetables
- It's a diverse field, with more and more interesting applications popping up every day.



Why Learn Computer Vision?

142

- ❑ Computer vision is built upon the fields of mathematics, physics, biology, engineering, and of course, computer science.
- ❑ There are many fields related to computer vision, such as machine learning, signal processing, robotics, and artificial intelligence.
- ❑ Even though it is a field built on advanced concepts, more and more tools are making it accessible to everyone from hobbyists to vision engineers to academic researchers.
- ❑ It is an exciting time in this field, and there are an endless number of possibilities for applications.
- ❑ One of the things that makes it exciting is that these days, the hardware requirements are inexpensive enough to allow more casual developers entry into the field, opening the door to many new applications and innovations.