Robot Vision and Image Processing

Week #9

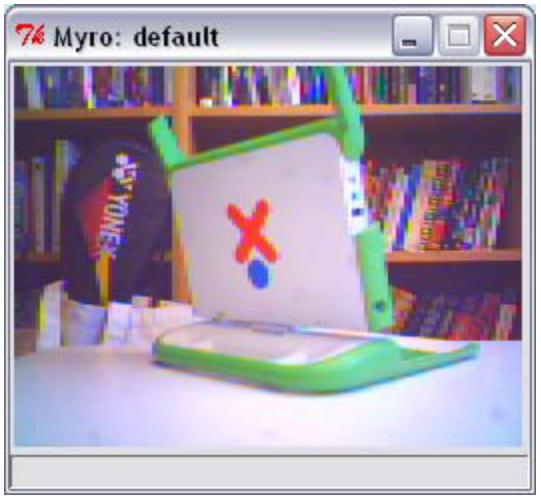
Prof. Ryan Kastner

Robot Vision

- The Scribbler has a small digital camera
- Pictures taken by the camera is called *image*
- We can perform computation on images
  - e.g. Face Detection

Robot Vision

- The image taken by a camera can serve as the eye of the Robot



pic = takePicture()
show(pic)

- In a color image, each pixel contains color information which is made up of the amount of red, green, and blue (also called, RGB) values
  - Values can be in the range [0..255]

Image

- A grayscale image contains only the level of gray in a pixel
  - Takes a single byte between 0 (white) and 255 (black)
    - How many bits in a byte?
- Image is just a 2-dimensional array of pixels
- Images obtained from the Scribbler have 256×192 (WxH)
- 49,152 pixels
- Each pixel is 3 bytes. So, scribbler images are 147,456 bytes in size

Pixels

- All digital cameras are sold by specifying the number of megapixels
- A camera is referred by the size of the largest image it can take
  - The Scribbler camera, has an image size of 147,456 bytes
  - It is only about 0.14 megapixels

UCSD

Saving Images

- Electronic storage and transfer can be made by compressing the data in the image
- Compressed Formats: JPEG, GIF, PNG etc.
- Scribbler supports JPEG and GIF
- Image Functions: Try this

```
picWidth = getWidth(pic)
picHeight = getHeight(pic)
print "Image WxH is", picWidth, "x", picHeight, "pixels."
```

Saving Images

savePicture(pic, "OfficeScene.jpg")

savePicture(pic, "OfficeScene.gif")

- Loading from disk:

  mySavedPicture = makePicture("OfficeScene.jpg")

  show(mySavedPicture)

- Try this:

  mySavedPicture = makePicture(pickAFile())

  show(mySavedPicture)

  Gives a navigational dialog box

- Click on the picture, what do you get?

Robot Explorer

- Taking gray-scale picture is faster than taking a color picture
  - You can update the images faster and also use as a camera
- Try this:

```
joyStick()
for i in range(25):
    pic = takePicture("gray")
    show(pic)
```

Movies

- The *savePicture()* function allows make animated GIF which in a browser shows several images one after the other

```
pic1 = takePicture()
turnLeft(0.5,0.25)
pic2 = takePicture()
turnLeft(0.5,0.25)
pic3 = takePicture()
turnLeft(0.5,0.25)
pic4 = takePicture()
listOfPictures = [pic1, pic2, pic3, pic4]
savePicture(listOfPictures, "turningMovie.gif")
```

Can you write this program using a for loop?

Making Pictures

- You can also make your own pictures
- Try this:

```
W = H = 100
newPic = makePicture(W, H, black)
show(newPic)
```

- Try this:

```
for x in range(W)
    for y in range(H):
        pixel = getPixel(newPic, x, y)
        setColor(pixel, white)
repaint(newPic)
```

Selecting Colors

- You can
  - Set a color: *setColor(pixel, white)*
  - Create a new color if you know the RGB values of the color: *myRed = makeColor(255, 0, 0)*
  - Visually select a color: *myColor = pickAColor()*
- The repaint command refreshes the displayed image:

```
for x in range(W)
    for y in range(H):
        pixel = getPixel(newPic, x, y)
        setColor(pixel, white)
repaint(newPic)
```

Do you see any problems?

Image Processing

- Way of taking existing images and transforming them in interesting ways
- You access an individual pixel and its color value, and transform it in any way you like
- Examples: Shrinking & Enlarging, Blurring & Sharpening, Negative & Embossing and Object Detection

Shrinking & Enlarging

- Write a program that will take an input image and shrink it by a factor, say F

- For example: if the original image is 3000×3000 pixels and we shrink it by a factor of 10, we would end up with an image of 300×300 pixels

*New pixel at x, y is a copy of the old pixel x*F, y*F*

Shrinking & Enlarging

```
def main():
    # read an image and display it
    oldPic = makePicture(pickAFile())
    show(myPic, "Before")

    X = getWidth(oldPic)
    Y = getHeight(oldPic)

    # Input the shrink factor and computer size of new image
    F = int(ask("Enter the shrink factor."))
    newx = X/F
    newy = Y/F

    # create the new image
    newPic = makePicture(newx, newy)

    for x in range(newx):
        for y in range(newy):
            setPixel(newPic, x, y, getPixel(myPic, x*F, y*F))
    show(newPic, "After")
```

Image Processing

- How does Scribbler recognize a ball?
- Once it recognizes it, can it follow the ball wherever it goes?

Image Processing

- To identify an object on an image, click on the image to get the RGB values
- Set the remaining pixels to 0
- Thus the robot can identify the object as in the image shown below
- Change the threshold values to get more refined identification

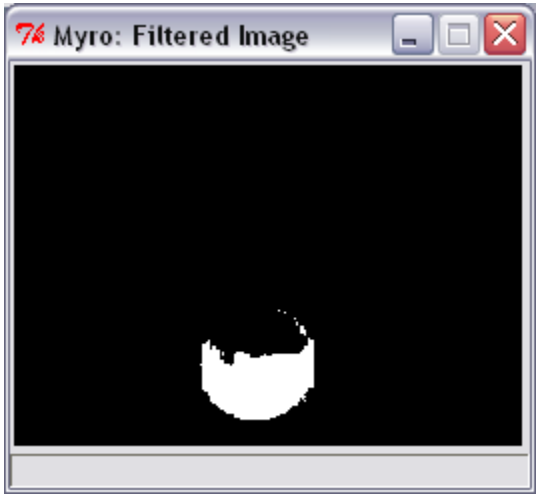
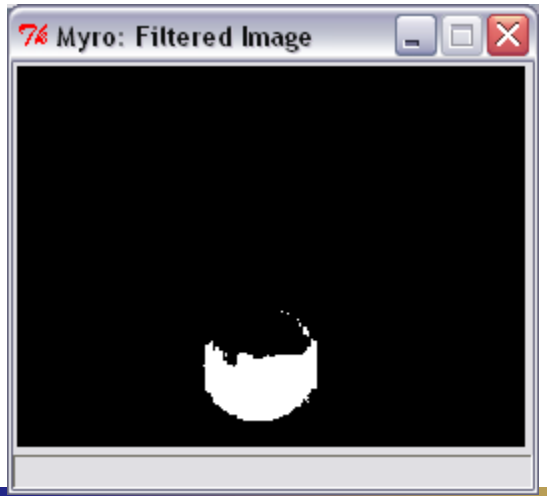


Image Processing

```
for pixel in getPixels(p):
        r, g, b = getRGB(pixel)
        if r > 200 and g < 100:
                setRGB(pixel, (255, 255, 255))
        else
                setRGB(pixel, (0, 0, 0))
```



UCSD

Image Processing

- Once you have identified the white image in the processes image, you can find the position of the object by taking the average of the *x*- locations
- What is the purpose of this function:

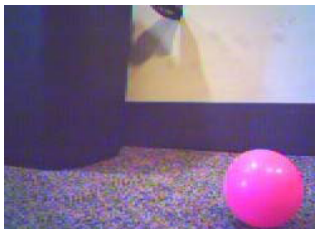
```
def ?(picture)
        tot_x = 0
        count = 0
        for pixel in getPixels(p):
                r, g, b = getRGB(pixel)
                if r > 200 and g < 100:
                        tot_x = tot_x + getX(pixel)
                        count = count + 1
        return tot_x/count
```

Image Processing

- You can use the *count* value and make the robot follow the movement of the ball

```
while timeTemaining(T):
        # take picture and locate the ball
        pic = takePicture()
        ballLocation = locateBall(pic)

        if ballLocation <= 85:
                turnLeft(cruiseSpeed)
        elif ballLocation <= 170:
                forward(cruiseSpeed)
        else:
                turnRight(cruiseSpeed)
```



Here ballLocation contains *count* value