91.451 Robotics II: Vision and Manipulation

Lecture 4
Spring 2005
Prof. Holly Yanco
Uses of Computer Vision: Surgical Systems

Articulated Arm
Laser-Calibrated Video Camera
Sun UltraSPARC Workstation
Flashpoint Controller
Laser Scanner Hardware

Laser on a Stepper Motor
Flashpoint Bar
Uses of Computer Vision: Content Based Image Retrieval

Sample Image

Retrieved Images
Uses of Computer Vision: Face Detection
Uses of Computer Vision: Car Tracking
Street Crossing

- Tracking from a moving platform
- Need to look left and right to find a safe time to cross
- Need to look ahead to drive to other side of road
- Must stay in crosswalk
Algorithm for Tracking Cars

1. Image differencing to find motion
2. Noise filtering using 3x3 median filter
3. Computation of Sobel edges
4. Use Mori’s sign pattern to find bottoms of cars [Mori 1994]
5. Find bounding boxes of moving objects
6. Use knowledge from prior frames to mark direction of travel of each bounding box
Getting Images

- Camera images grabbed with a framegrabber (on desktop or on robot)
- Images can be in grayscale or color (RGB or HSV)
- Grayscale is a single value from 0 (black) to 255 (white)
- Most vision processing now done with color images
RGB Color Space

- Lighting impacts color values!
HSV Color Space

• Hue, the color type (such as red, blue, or yellow);
  – Measured in values of 0-360 by the central tendency of its wavelength

• Saturation, the 'intensity' of the color (or how much grayness is present),
  – Measured in values of 0-100% by the amplitude of the wavelength

• Value, the brightness of the color.
  – Measured in values of 0-100% by the spread of the wavelength
HSV Color Space

- Other pictorial representations:
Looking for Colors

• Can train on colors in a region of the image, then track that color

• Best to track colors in HSV color space (RGB is too lighting dependent)
Image Processing

• Grab image
• Filter to smooth image
• Process for some property
  – Intensity changes for edges
  – Blobbing to find an area of a particular color
• Act on the results
Filtering Methods

- Median
- Mean
- Gaussian
Gaussian filter

- This filter $H$ is a good approximation to $h(u, v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{\sigma^2}}$
- Properties of Gaussian
  - more weight to the center
  - good model of blurring in optical systems
  - $\sigma$ corresponds to width of the Gaussian
Gaussian Filter
Mean Blur

- Flurs the image by changing the color of the pixel being looked at to the mean value of the pixels surrounding it. The number of surrounding pixels being looked at is defined by the kernel parameter. If kernel is 3, then the pixel being looked at is the center of a 3x3 box, shown in the diagram.
Mean Blur
Median Blur

- Blurs the image by changing the color of the pixel being looked at to the median value of the pixels surrounding it. The number of surrounding pixels being looked at is defined by the kernel parameter. If kernel is 3, then the pixel being looked at is the center of a 3x3 box, shown in the diagram.

![Diagram showing median blur process]
Median Blur
Edge Detection: Sobel

\[ |G| = \sqrt{Gx^2 + Gy^2} \]
Edge Detection: Sobel
Edge Detection: Canny

1. Apply Gaussian filter
2. Sobel edge detection
3. Find direction of edges
4. Relate edge direction to direction that can be traced in an image
5. Nonmaximum suppression used to trace along the edge in the edge direction to suppress any pixel value that is not considered to be an edge
6. Hysteresis used to eliminate streaking (breaking up of an edge contour)
Edge Detection: Canny
Phission

• For vision processing, we’ll use a system called Phission, developed by Phil Thoren
• Written in C++
• SWIG used to create Python calls
• Will use filters from phissioncamera for this week’s lab
Phission: Random Filter Example

• Following code shows how to call Phission functions using phissioncamera

• This direct control brain generates a random number, then calls a filter to run for 50 time steps
from pyro.brain import Brain
from time import *
from random import *
from phissioncamera import *

class SimpleBrain(Brain):
    def setup(self):
        self.counter = 0          # initialize counter to 0
    def step(self):
        self.counter += 1        # increment counter
        if ((self.counter % 50) == 0):  # run randomFilter() every 50 iterations
            self.randomFilter()

    def randomFilter(self):
        x = randrange(0,7,1)    # get a random number
        if(x == 0):
            self.camera.canny_Filter()
            print "canny_Filter()"
        if(x == 1):
            self.camera.sobel_Filter()
            print "sobel_Filter()"
        if(x == 2):
            self.camera.gaussianBlur_Filter()
            print "gaussianBlur_Filter()"
        if(x == 3):
            self.camera.medianBlur_Filter()
            print "medianBlur_Filter()"
        if(x == 4):
            self.camera.motion_Filter()
            print "motion_Filter()"
        if(x == 5):
            self.camera.SpotColorTrackHSV_Filter(160,120,12, (12,60,120) )
            print "SpotColorTrackHSV_Filter(160,120,12, (12,60,120) )"
        if(x == 6):
            self.camera.SpotColorTrackRGB_Filter(160,120,12, (30,30,30) )
            print "SpotColorTrackRGB_Filter(160,120,12, (30,30,30) )"
def INIT(engine):
    camera = phissioncamera()  # create a new phission camera
    camera.setup(1,320,240)    # set up the phission camera -- change first param to 0 if not on robot
    brain = SimpleBrain('SimpleBrain', engine)  # create the brain
    brain.camera = camera      # set the brain's camera variable
    return brain
This Week’s Lab

- Using the Phission vision functions to find, track and pick up an object of a particular color
- Will use gripper
- Will use sequencing (fsm) based control
Sequencing (FSM) Control

approachPuck → locatePuck

locatePuck → grabPuck

grabPuck → done
from pyro.brain.behaviors.fsm import *
from time import *

class GatherPucksBrain(FSMBrain):
    def setup(self):
        self.robot.startDevice('blobfinder')
        self.robot.startDevice('gripper')

    def destroy(self):
        self.robot.removeDevice('blobfinder0')
        self.robot.removeDevice('gripper0')

class locatePuck(State):
    """
    Rotate until a red blob is seen or a complete scan has been done.
    """
    def onActivate(self):
        self.searches = 0  # counter for number of rotations

    def update(self):
        redBlobs = self.robot.get('robot/blobfinder/data')[1][0]
        if len(redBlobs) != 0:
            self.robot.move(0, 0)
            print "found a puck!"
            self.goto('approachPuck')
        elif self.searches > 275:
            print "found all pucks"
            self.goto('done')
        else:
            print "searching for a puck"
            self.searches += 1  # update counter
            self.robot.move(0, 0.2)
class approachPuck(State):
    ""
    Move towards closest red blob by centering it in blobfinder. Once centered, then consider distance to the closest red blob. Once close, move slowly until it is within gripper.
    ""
    def update(self):
        redBlobs = self.robot.get('robot/blobfinder/data')[1][0]
        if len(redBlobs) == 0:
            print "no puck in sight"
            self.goto('locatePuck')
        elif redBlobs[0][2] < 50:
            print "puck far left"
            self.robot.move(0, 0.2)
        elif redBlobs[0][2] < 75:
            print "puck to left"
            self.robot.move(0.2, 0.1)
        elif redBlobs[0][2] > 100:
            print "puck far right"
            self.robot.move(0, -0.2)
        elif redBlobs[0][2] > 85:
            print "puck to right"
            self.robot.move(0.2, -0.1)
        else:
            if self.robot.get('robot/gripper/breakBeamState') > 1:
                print "grabbing puck"
                self.robot.move(0, 0)
            self.goto('grabPuck')
        elif redBlobs[0][8] > 500:
            print "puck straight ahead"
            self.robot.move(0.3, 0)
        else:
            print "puck very close"
            self.robot.move(0.1, 0)
class grabPuck(State):
    """
    Pick up the current puck and then return to locating another puck.
    """
    def update(self):
        self.robot.set('robot/gripper/command', 'close')
        sleep(1.0)
        self.robot.set('robot/gripper/command', 'open')
        sleep(1.0)
        self.goto('locatePuck')

class done(State):
    """
    No more visible pucks, so stop moving.
    """
    def update(self):
        self.robot.move(0, 0)

def INIT(engine):
    brain = GatherPucksBrain(engine)
    brain.add(locatePuck(1))
    brain.add(approachPuck())
    brain.add(grabPuck())
    brain.add(done())
    return brain
Pan-Tilt-Zoom

This device is used to control a pan-tilt-zoom camera's actuator modes. To start this device do:

```python
>>> robot.startDevice('ptz')
Loading device 'ptz0'...
ptz0
```

You can examine the attributes of the device as shown below:

```python
>>> robot.get('robot/ptz')
['command', 'data', 'name', 'pan', 'pose', 'tilt', 'type', 'zoom']
```
Pan-Tilt-Zoom

The current state of the ptz device can be obtained by accessing the `pose` attribute:

```python
>>> robot.get('robot/ptz/pose')
(0, 0, 60, 0, 0)
```

The numbers reported are the current values of pan, tilt, and zoom (or field of view) in degrees, followed by the pan and tilt velocities in degrees/second. All of these can be individually set using the `set` method on the `command` attribute:

```python
set('/robot/ptz/COMMAND', VALUE)
```

where `COMMAND` is: `pose`, `pan`, `tilt`, `zoom`. 
Using the Gripper

```python
>>> robot.startDevice('gripper')
Loading device 'gripper0'...
gripper0
```
Gripper methods

```python
>> robot.get('robot/gripper')
['breakBeamState', 'command', 'isClosed', 'isLiftMaxed', 'isLiftMoving', 'isMoving', 'name', 'state', 'type']
```
Using the Gripper

The general syntax for setting different commands is:

```
robot.set('robot/gripper/command', VALUE)
```

where `VALUE` is one of the options listed below.

There are several specific values that can be set for the `command` attribute. These correspond to various actuator commands for the gripper. These are listed below:

- `'open'` - opens the paddles of the gripper.
- `'close'` - closes the paddles of the gripper.
- `'stop'` - stops the paddles if they are in motion.
- `'up'` - raises the paddles up.
- `'down'` - lowers the paddles down.
- `'store'` - raises and closes the paddles.
- `'deploy'` - lowers and opens the paddles.
- `'halt'` - stops both horizontal and vertical movement of the grippers.
Using the Gripper

To lower the gripper onto something, you will position the robot so that the gripper is open and positioned above the object. Then you would lower the gripper, close it, and lift it up:

```python
>>> robot.set('robot/gripper/command', 'down')
Ok
>>> robot.set('robot/gripper/command', 'close')
Ok
>>> robot.set('robot/gripper/command', 'up')
Ok
```

Or, alternately, you could:

```python
>>> robot.set('robot/gripper/command', 'deploy')
Ok
>>> robot.set('robot/gripper/command', 'store')
Ok
```
More gripper attributes

States of the gripper are accessible through the state and breakBeamState attributes. When queried, you will get a number returned which should be interpreted as shown below:

Gripper States
  – 0 if paddles are moving.
  – 1 if paddles are open.
  – 2 if paddles are closed.

Break Beam States - finds out if something is between the gripper paddles.
  – 0 if neither beam is broken.
  – 1 if back beam is broken.
  – 2 if front beam is broken.
  – 3 if both beams are broken.
In Lab This Week

• You can try the Phission functions on your desktop machine using the camera (plug it in while using and unplug when done)
• You can try the sequencing example in Stage
• You will write code to find and pick up an object on the robot using Phission (need to test full program on robot, but could test parts that don’t rely on vision in the simulator)