OpenCV

Gary Bradski
Senior Scientist, Willow Garage
Consulting Prof. Stanford University

Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

OpenCV Overview:

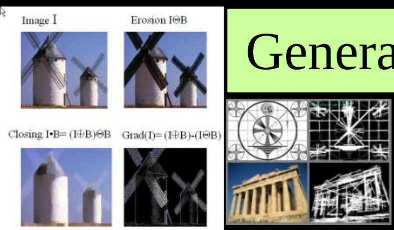
Willow Garage

opencv.willowgarage.com

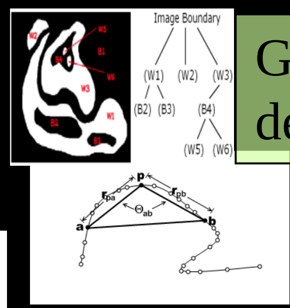
> 500 algorithms

Robot support

General Image Processing Functions



Geometric descriptors



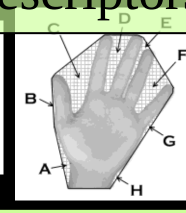
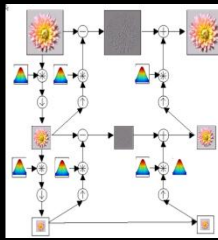
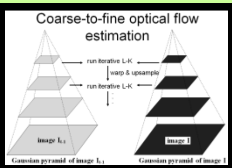
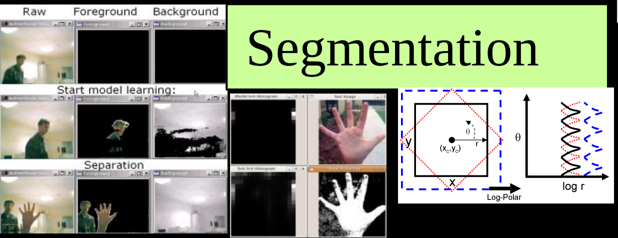


Image Pyramids

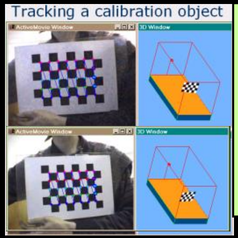


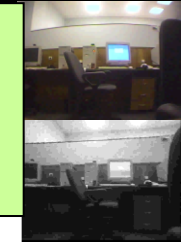


Segmentation

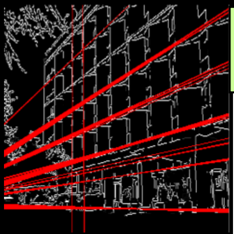


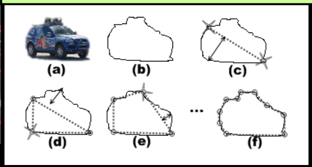
Camera calibration, Stereo, 3D



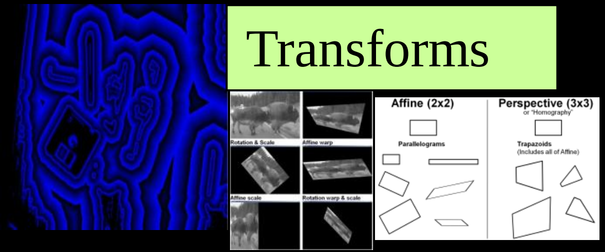


Features

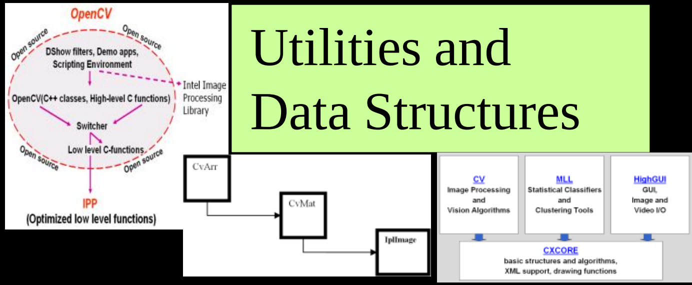




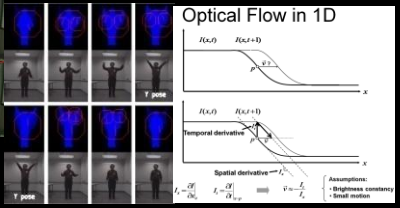
Transforms



Utilities and Data Structures

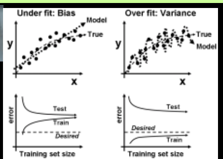


Tracking

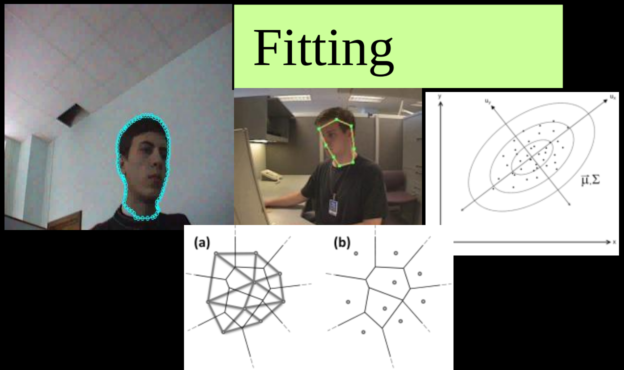


Machine Learning:

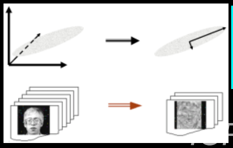
- Detection,
- Recognition



Fitting



Matrix Math

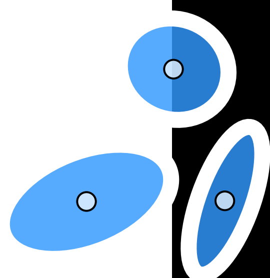


OpenCV

Machine Learning Library (MLL)

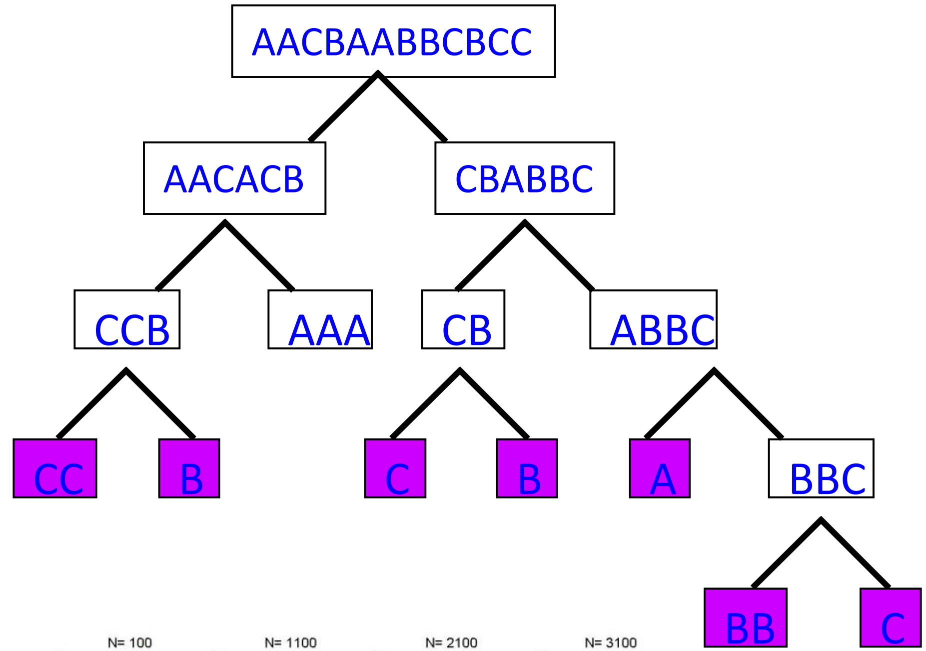
CLASSIFICATION / REGRESSION
Fast Approximate NN (FLANN)
Extremely Random Trees
Random Forests
Statistical Boosting, 4 flavors
CART
Naïve Bayes
MLP (Back propagation)
SVM
Face Detector
(Histogram matching)
(Correlation)

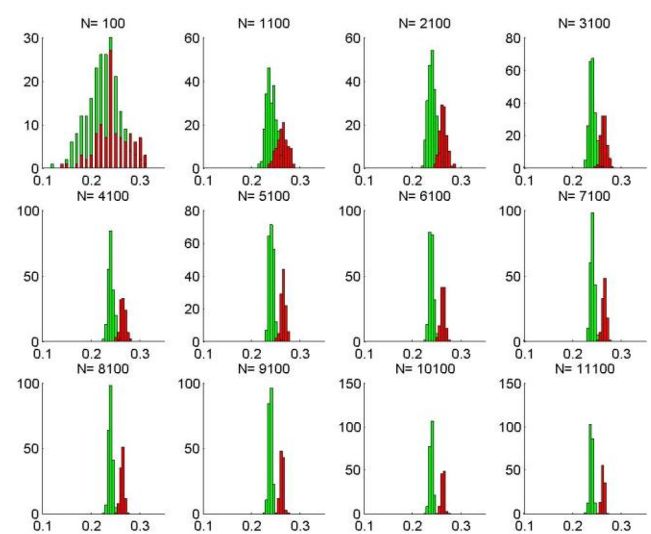
CLUSTERING
K-Means
EM
(Mahalanobis distance)



TUNING/VALIDATION
Cross validation
Bootstrapping
Variable importance
Sampling methods

http://opencv.willowgarage.com





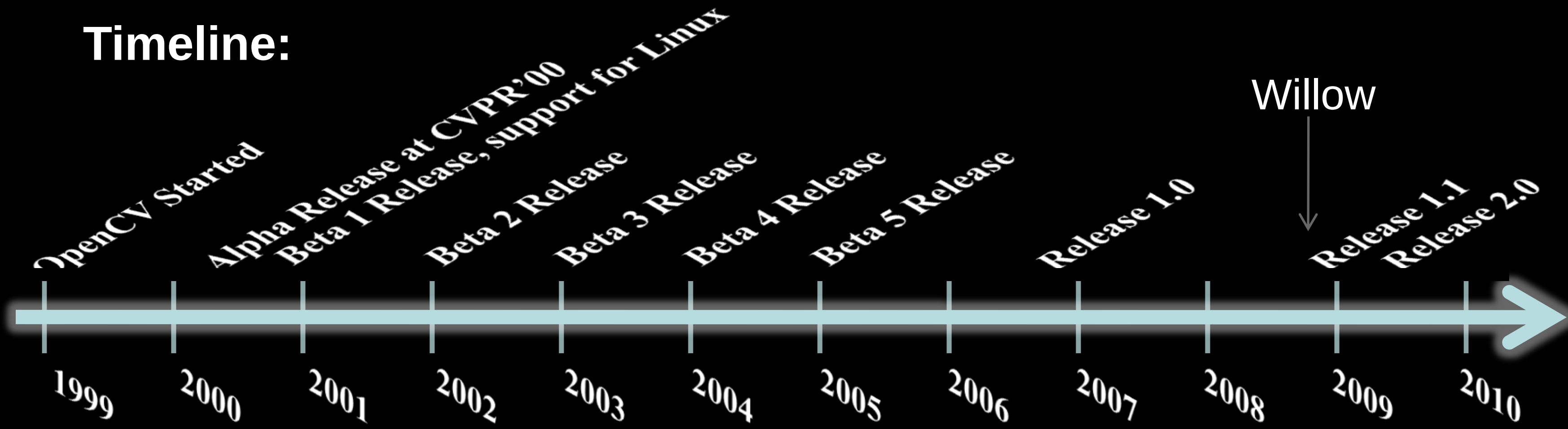
OpenCV History

Original goal:

Accelerate the field by lowering the bar to computer vision

Find compelling uses for the increasing MIPS out in the market

Timeline:



Staffing:

Climbed in 1999 to average 7 first couple of years

Starting 2003 support declined between zero and one with exception of transferring the machine learning from manufacturing work I led (equivalent of 3 people).

Support to zero the couple of years before Willow.

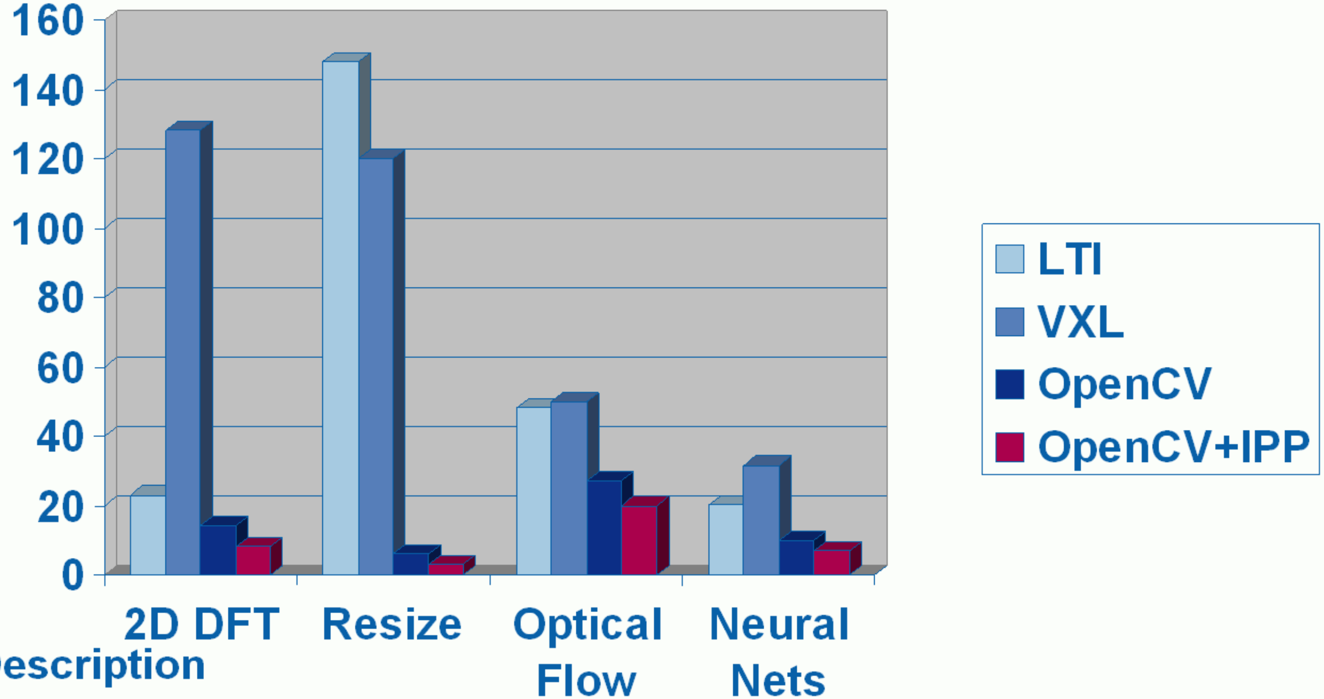
5 people last year, 7 people this year

OpenCV License

- OpenCV has a BSD license
  - It is free and open for
    - Commercial and Research use
    - In whole or in part
  - Supported by:
    - Willow Garage (5 Developers)
    - Nvidia (2 Developers, CUDA)
    - Intel (Test standards)
    - Google (8 Summer of Code Interns)
    - Community (User group

OpenCV Tends Towards Real Time

Comparison with other libs: Performance



Description

Test station: Pentium M, 1.7GHz

Libraries: OpenCV 1.0pre, IPP 5.0, LTI 1.9.14, VXL 1.4.0

2D DFT: Forward Fourier Transform of 512x512 image

Resize: 512x512->384x384 bilinear interpolation, 8-bit 3-channel image

Optical flow: 520 points tracked with 41x41 window, 4 pyramid levels.

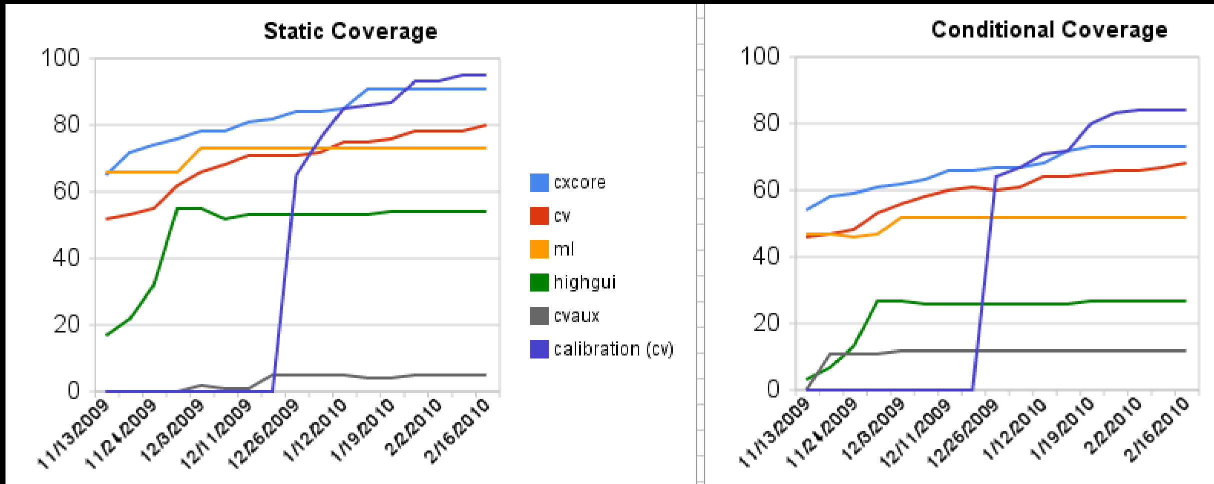
Neural Net: mushroom benchmark from FANN

http://opencv.willowgarage.com

Willow Garage

Software Engineering

- Works on:
  - Linux, Windows, Mac OS
- Languages:
  - C++, Python, C
- Online documentation:
  - Online reference manuals: C++, C and Python.
- We've been expanding Unit test code
- Will soon standarize on cxx or Google's test system.



OpenCV Directory Structure

```
opencv/
    include/opencv/              # headers for backward compatibility
    modules/

            # decomposed cxcore, cv, highgui, ml, cvaux:

            core/                # core functionality
            imgproc/             # image processing part of cv
            highgui/             # GUI
            ml/                  # machine learning routines
            video/               # optical flow, motion templates, Kalman filter,
                                   blob tracking, background/foreground segmentation
            calib3d/             # camera calibration, epipolar geometry,
                                   stereo correspondence
            features2d/          # 2d feature detectors & descriptors
            objdetect/           # Haar/LBP & HOG object detectors,
                                   fern-based planar object detector
            legacy/              # obsolete functionality
            user_contrib/        # user contrib

            # some other stuff

            python               # new-style Python interface
            ffmpeg               # ffmpeg bindings
            traincascade         # Haar/LBP training application

     samples
     doc
     ...
```

New C++ API: Usage Example

Focus Detector

C:

```
double calcGradients(const IplImage *src, int aperture_size = 7)
{
    CvSize sz = cvGetSize(src);
      IplImage* img16_x = cvCreateImage( sz, IPL_DEPTH_16S, 1);
      IplImage* img16_y = cvCreateImage( sz, IPL_DEPTH_16S, 1);

      cvSobel( src, img16_x, 1, 0, aperture_size);
      cvSobel( src, img16_y, 0, 1, aperture_size);

    IplImage* imgF_x = cvCreateImage( sz, IPL_DEPTH_32F, 1);
    IplImage* imgF_y = cvCreateImage( sz, IPL_DEPTH_32F, 1);

    cvScale(img16_x, imgF_x);
    cvScale(img16_y, imgF_y);

    IplImage* magnitude = cvCreateImage( sz, IPL_DEPTH_32F, 1);
    cvCartToPolar(imgF_x, imgF_y, magnitude);
    double res = cvSum(magnitude).val[0];

    cvReleaseImage( &magnitude );
    cvReleaseImage(&imgF_x);
      cvReleaseImage(&imgF_y);
      cvReleaseImage(&img16_x);
      cvReleaseImage(&img16_y);

    return res;
}
```

C++:

```
double contrast_measure(const Mat& img)
{
  Mat dx, dy;

  Sobel(img, dx, 1, 0, 3, CV_32F);
  Sobel(img, dy, 0, 1, 3, CV_32F);
  magnitude(dx, dy, dx);

  return sum(dx)[0];
}
```

Python Face Detector Node: 1

The Setup

```
#!/usr/bin/python
"""
This program is demonstration  python ROS Node for face and object detection using haar-like features.
The program finds faces in a camera image or video stream and displays a red box around them.   Python
implementation by: Roman Stanchak, James Bowman
"""
import roslib
roslib.load_manifest('opencv_tests')
import sys
import os
from optparse import OptionParser
import rospy
import sensor_msgs.msg
from cv_bridge import CvBridge
import cv

# Parameters for haar detection
# From the API:
# The default parameters (scale_factor=2, min_neighbors=3, flags=0) are tuned
# for accurate yet slow object detection. For a faster operation on real video
# images the settings are:
# scale_factor=1.2, min_neighbors=2, flags=CV_HAAR_DO_CANNY_PRUNING,
# min_size=<minimum possible face size

min_size = (20, 20)
image_scale = 2
haar_scale = 1.2
min_neighbors = 2
haar_flags = 0
```

Python Face Detector Node: 2

The Core

```
if __name__ == '__main__':

    pkgdir = roslib.packages.get_pkg_dir("opencv2")
    haarfile = os.path.join(pkgdir, "opencv/share/opencv/haarcascades/haarcascade_frontalface_alt.xml")

    parser = OptionParser(usage = "usage: %prog [options] [filename|camera_index]")
    parser.add_option("-c", "--cascade", action="store", dest="cascade", type="str", help="Haar cascade file, default %default", default = haarfile)
    (options, args) = parser.parse_args()

    cascade = cv.Load(options.cascade)
    br = CvBridge()

    def detect_and_draw(imgmsg):
        img = br.imgmsg_to_cv(imgmsg, "bgr8")
        # allocate temporary images
        gray = cv.CreateImage((img.width,img.height), 8, 1)
        small_img = cv.CreateImage((cv.Round(img.width / image_scale),
                    cv.Round (img.height / image_scale)), 8, 1)

        # convert color input image to grayscale
        cv.CvtColor(img, gray, cv.CV_BGR2GRAY)

        # scale input image for faster processing
        cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)

        cv.EqualizeHist(small_img, small_img)

        if(cascade):
            faces = cv.HaarDetectObjects(small_img, cascade, cv.CreateMemStorage(0),
                                         haar_scale, min_neighbors, haar_flags, min_size)
            if faces:
                for ((x, y, w, h), n) in faces:
                    # the input to cv.HaarDetectObjects was resized, so scale the
                    # bounding box of each face and convert it to two CvPoints
                    pt1 = (int(x * image_scale), int(y * image_scale))
                    pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
                    cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8, 0)

        cv.ShowImage("result", img)
        cv.WaitKey(6)

    rospy.init_node('rosfacedetect')
    image_topic = rospy.resolve_name("image")
    rospy.Subscriber(image_topic, sensor_msgs.msg.Image, detect_and_draw)
    rospy.spin()
```

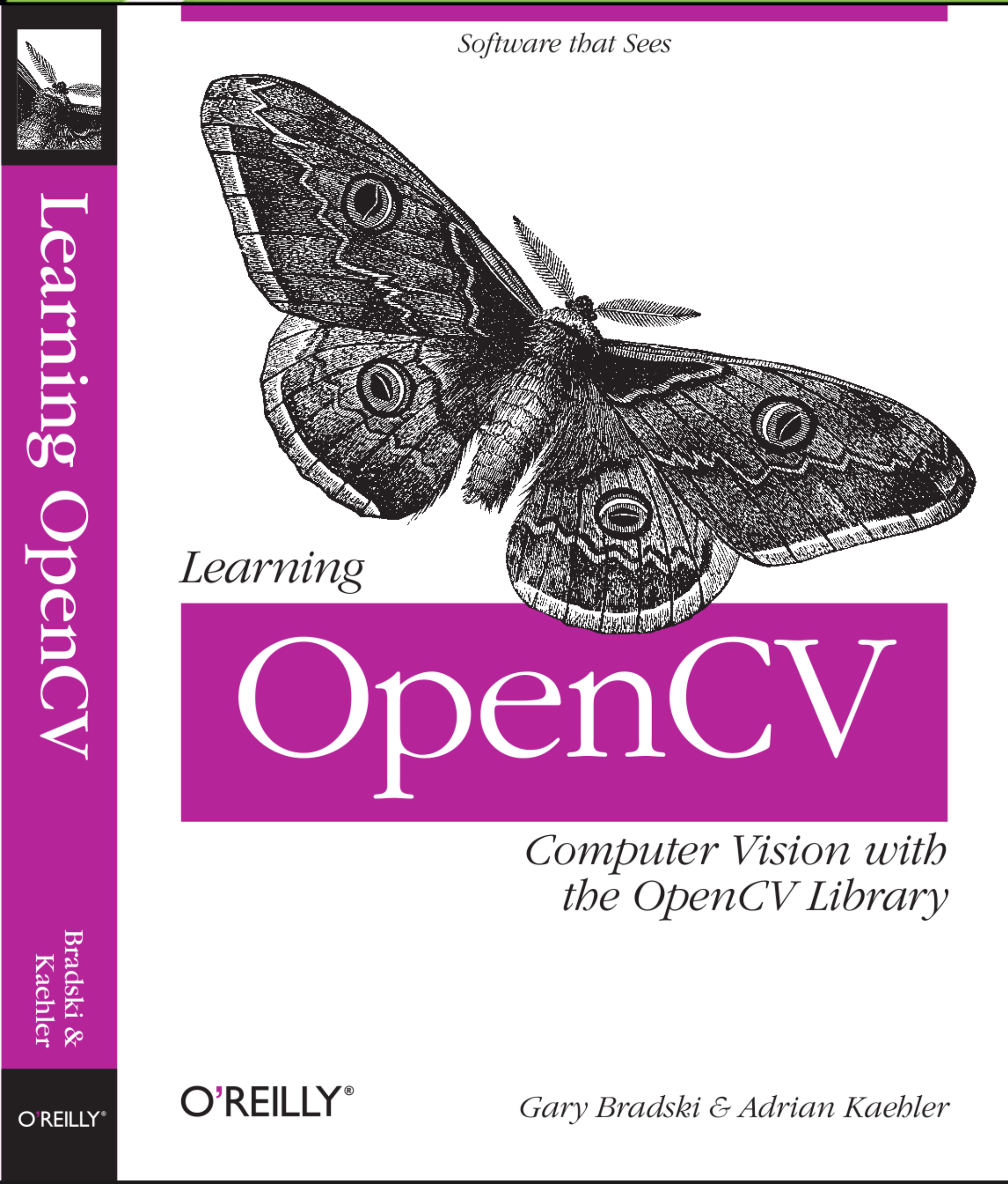
OpenCV Important Links

- Main site
  - http://opencv.willowgarage.com
- User site
  - http://opencv.willowgarage.com/wiki/FullOpenCVWiki
- User group (42000 members)
  - http://tech.groups.yahoo.com/group/OpenCV/
- Download/Install
  - http://opencv.willowgarage.com/wiki/InstallGuide
- OpenCV Book:
  - http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134
- Developer meeting notes
  - http://pr.willowgarage.com/wiki/OpenCVMeetingNotes

2010 Plans

- Re-org into coherent processing “stacks”
  - Texture and patch based object recognition
  - Visual Odometry and VSLAM
  - Stereo
  - Image stitching
  - 3D model capture
  - User contrib

OpenCV Book:



Learning OpenCV

- Computer Vision theory and practice

- http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134

Japanese and Chinese translations of the book are now available.

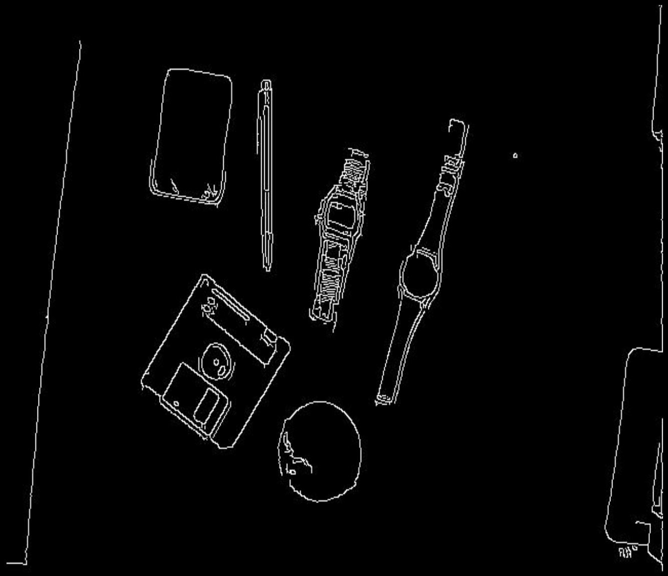
Summary

- Comprehensive computer vision and ML library
- BSD license, free for commercial or research
- C++, C, Python, Linux, Windows, Mac
- Re-orging into processing stacks

Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

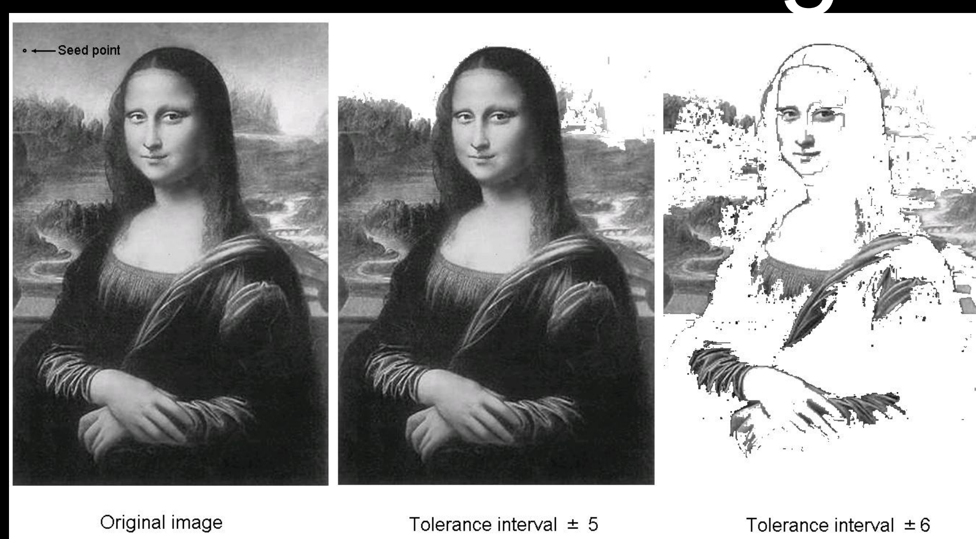
Canny Edge Detector



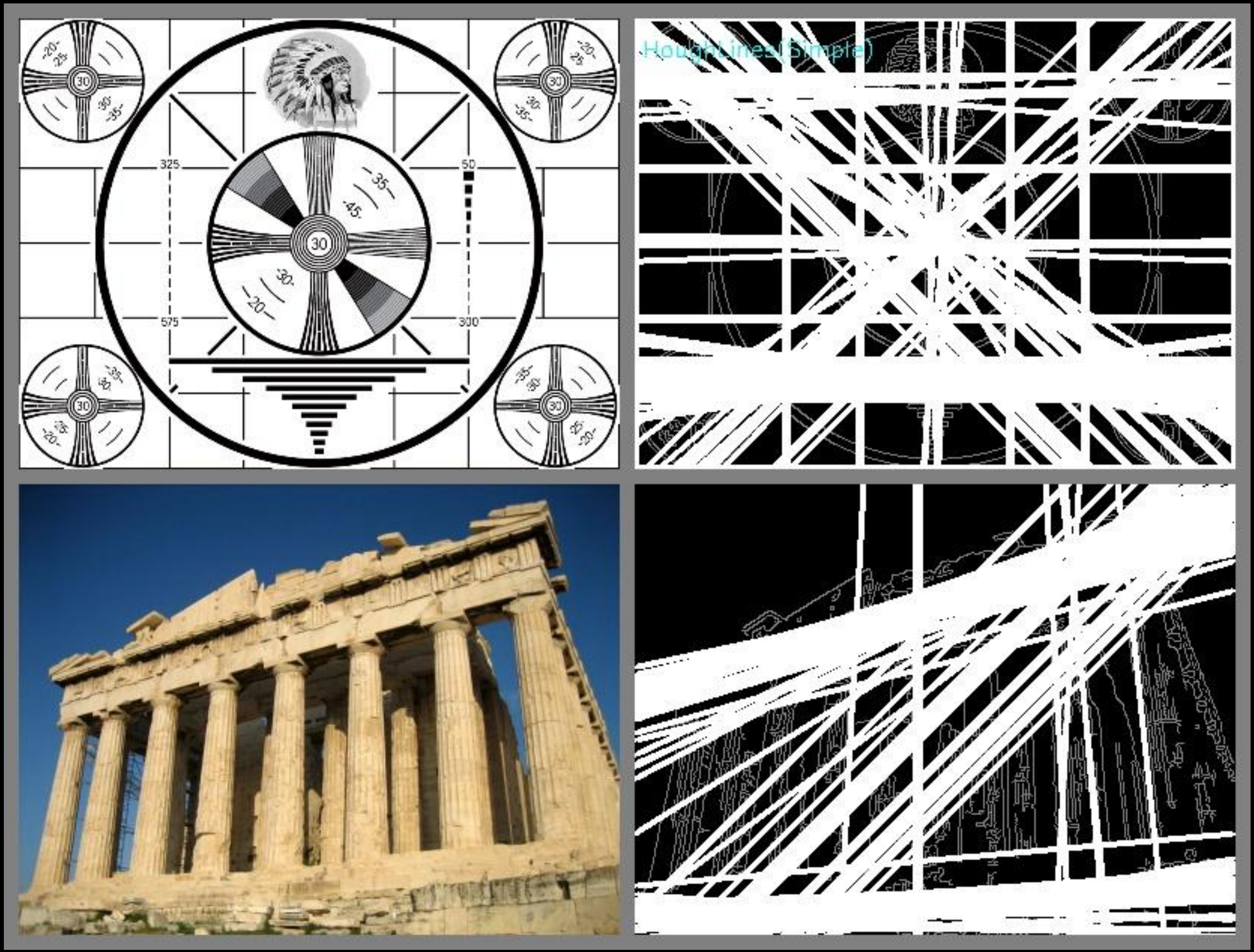
Distance Transform

- Distance field from edges of objects

Flood Filling



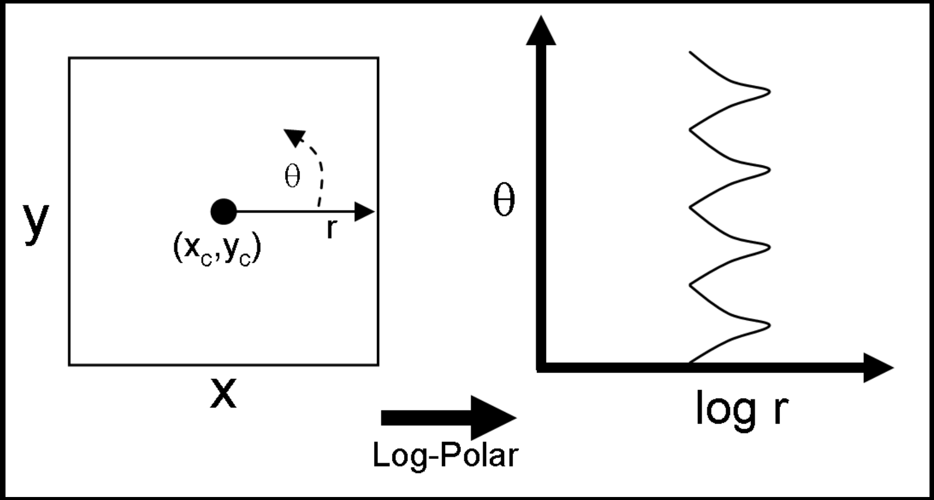
Hough Transform

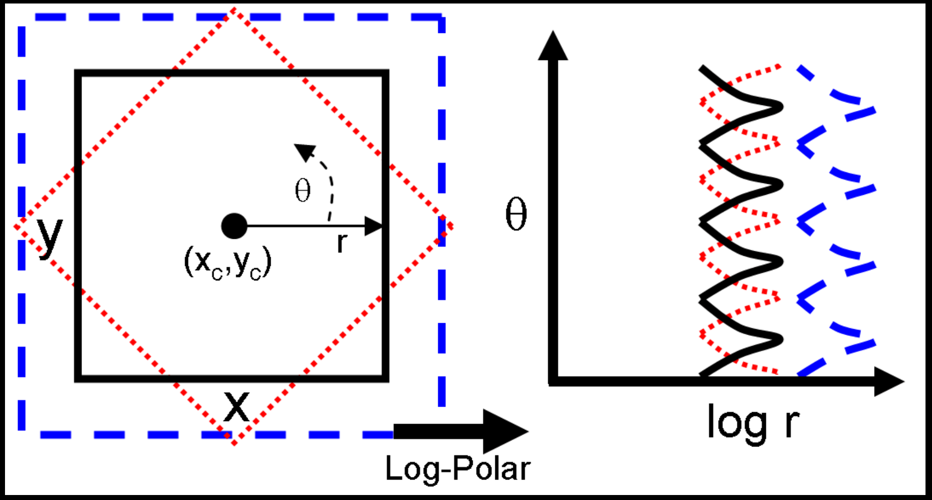


Gary Bradski, Adrian Kahler 2008

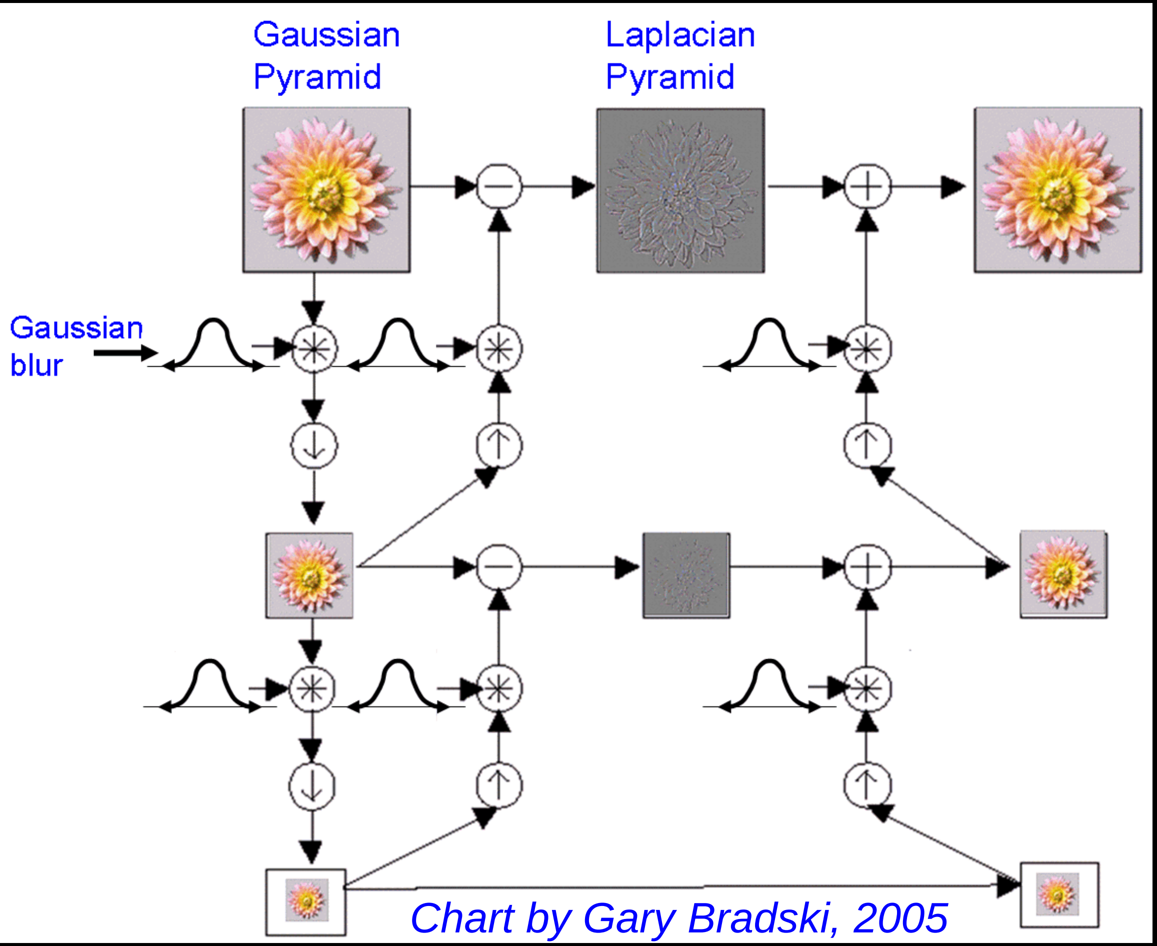
Space Variant vision: Log-Polar Transform

Screen shots by Gary Bradski, 2005





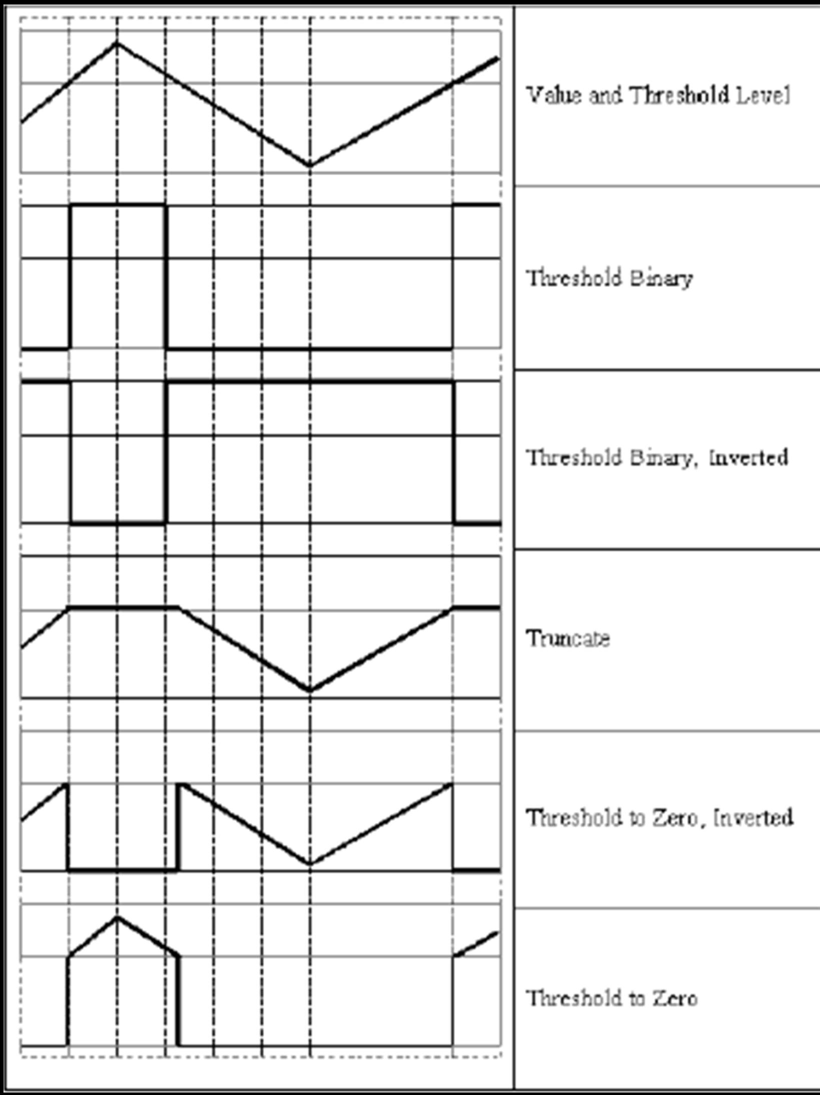
Scale Space

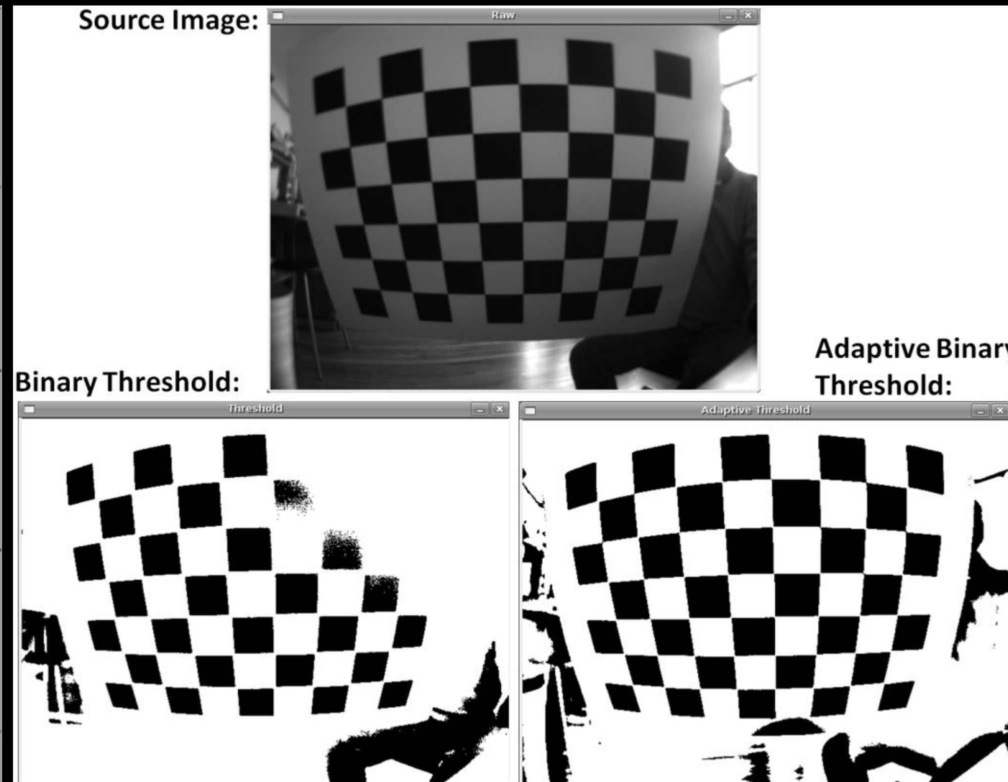



```
void cvPyrDown(
    IplImage* src,
    IplImage* dst,
    IplFilter    filter = IPL_GAUSSIAN_5x5);
```

```
void cvPyrUp(
    IplImage* src,
    IplImage* dst,
    IplFilter    filter = IPL_GAUSSIAN_5x5);
```

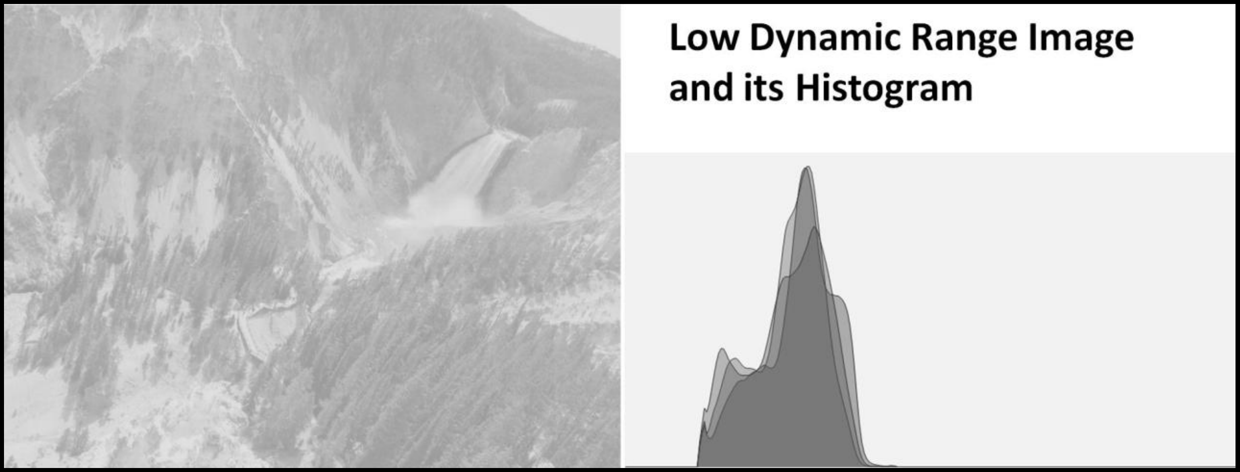
Thresholds

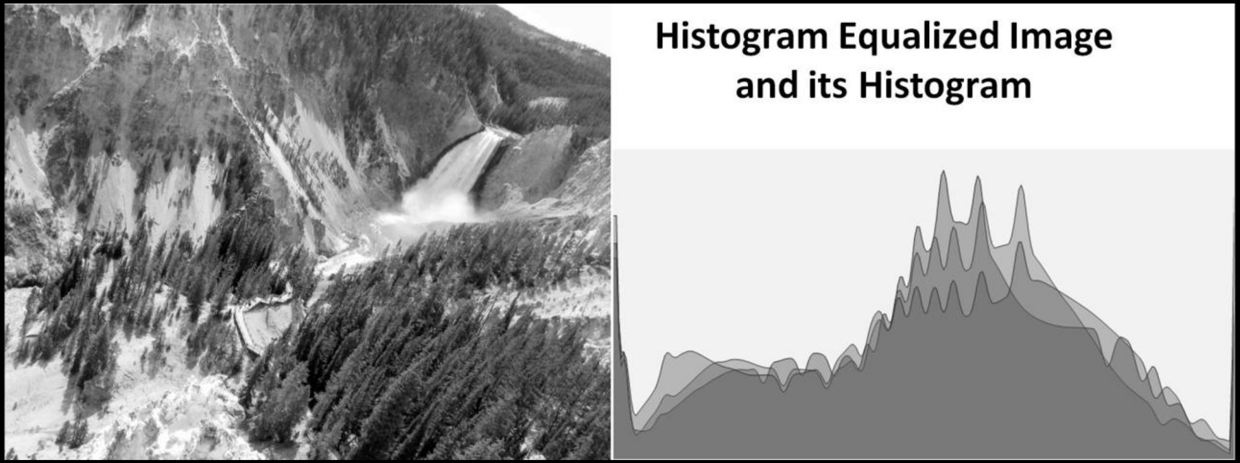




Screen shots by Gary Bradski, 2005

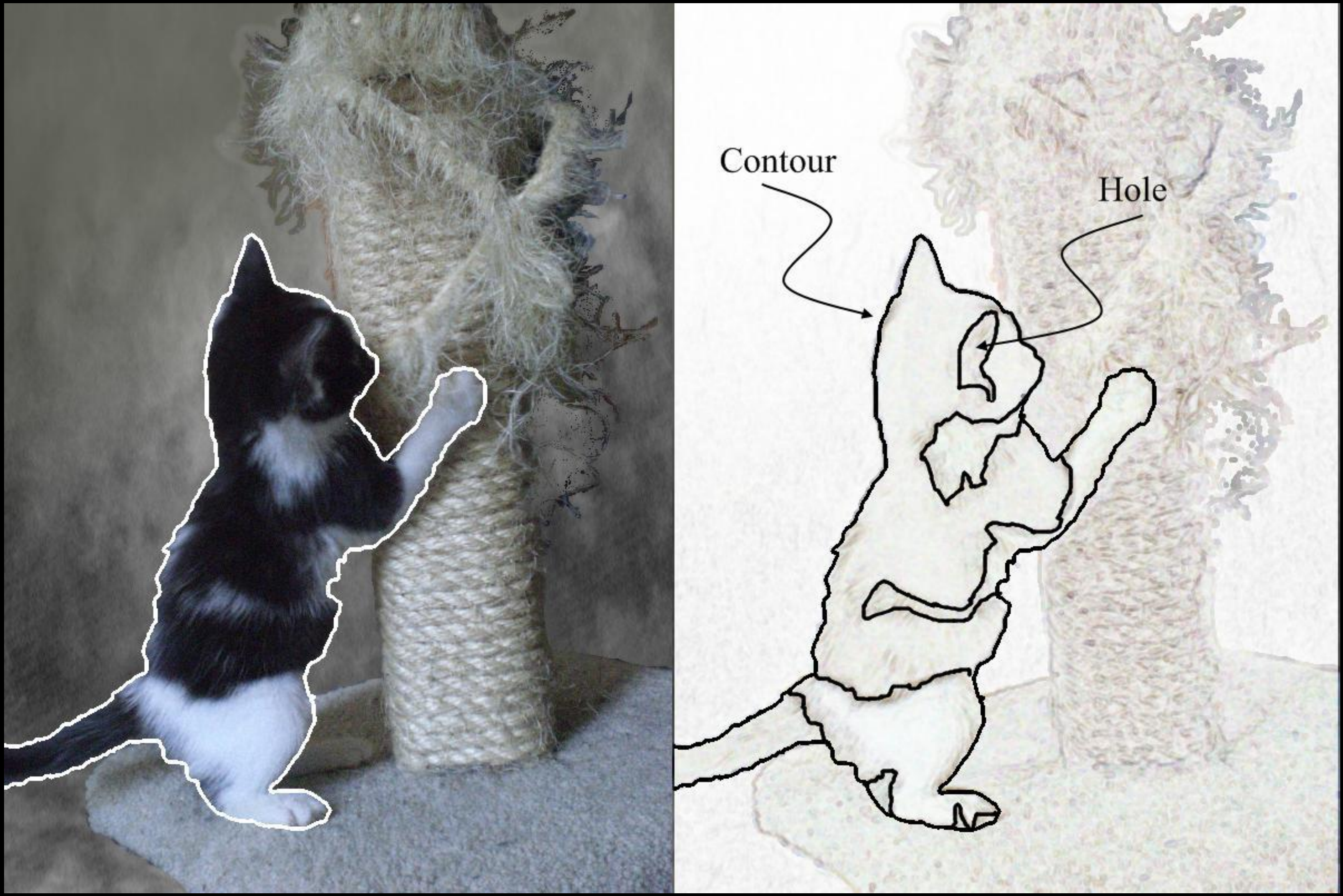
Histogram Equalization





Screen shots by Gary Bradski, 2005

Contours



Morphological Operations Examples

- Morphology - applying Min-Max. Filters and its combinations

Image I

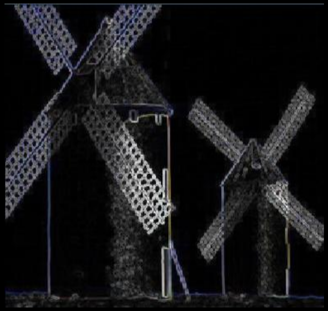
Erosion $I \Theta B$

Dilatation $I \oplus B$

Opening $I \circ B = (I \Theta B) \oplus B$

Closing $I \bullet B = (I \oplus B) \Theta B$

$\text{Grad}(I) = (I \oplus B) - (I \Theta B)$

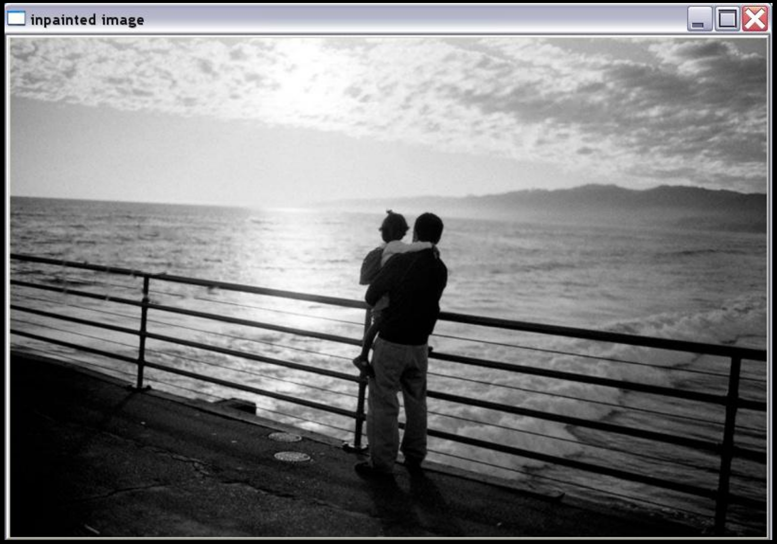


$\text{TopHat}(I) = I - (I \Theta B)$

$\text{BlackHat}(I) = (I \oplus B) - I$

Image textures

- Inpainting:
- Removes damage to images, in this case, it removes the text.



Segmentation

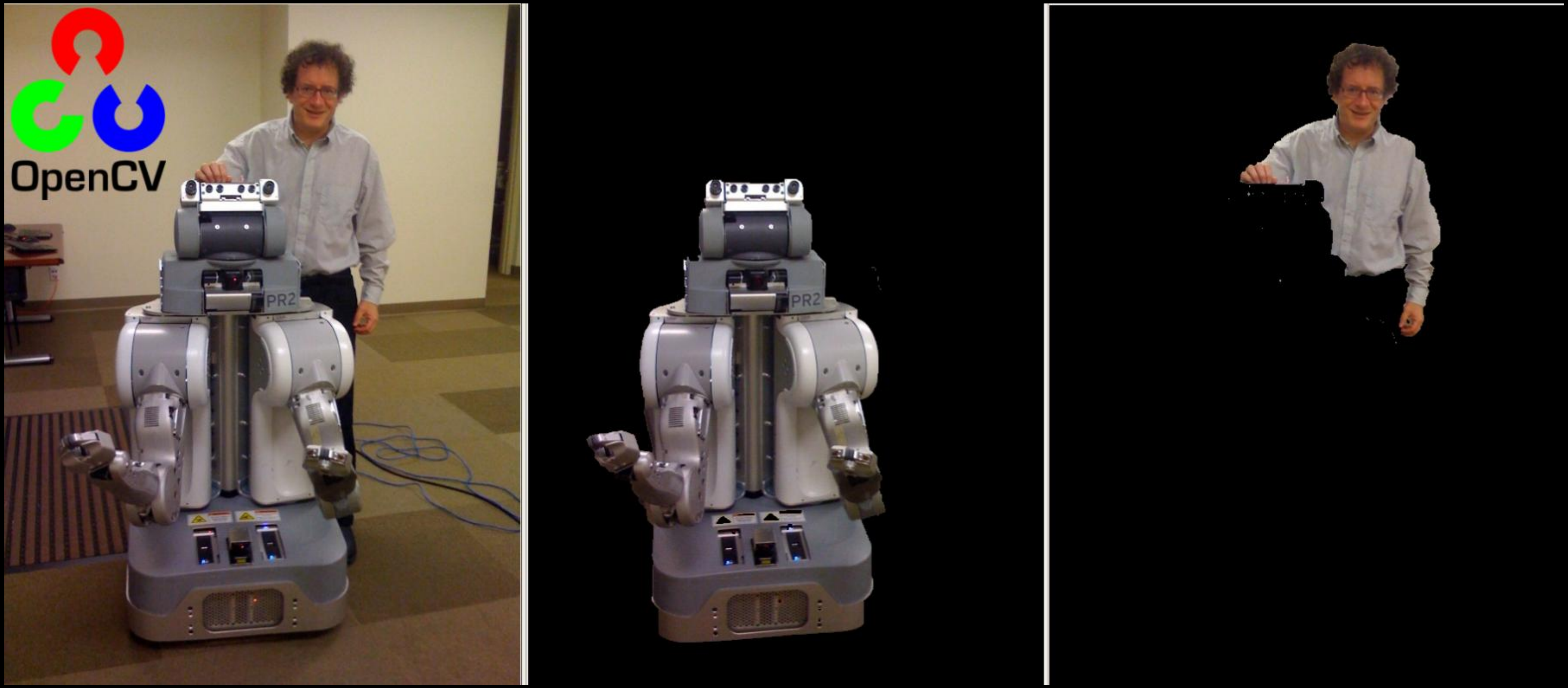
mean-shift, graph-cut

Here: Watershed

Screen shots by Gary Bradski, 2005

Recent Algorithms: GrabCut

- Graph Cut based segmentation



Images by Gary Bradski, © 2010

Motion Templates (work with James Davies)

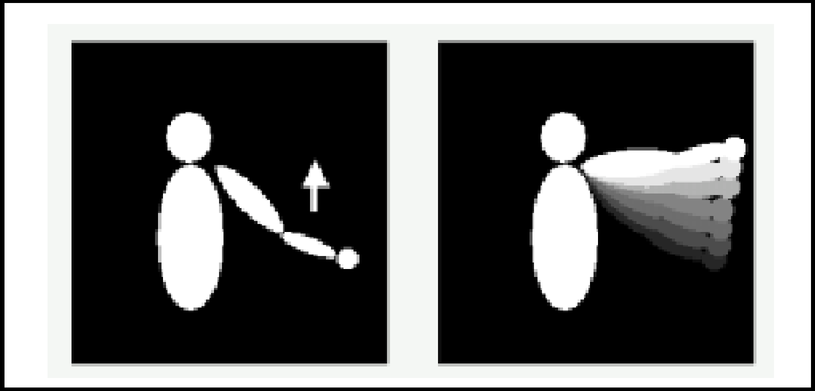
Object silhouette

Motion history images

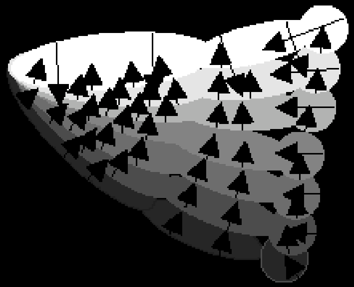
Motion history gradients

Motion segmentation algorithm

silhouette MHI



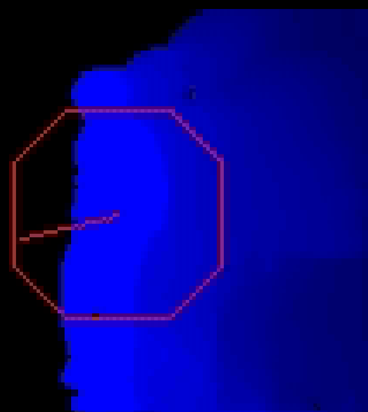
MHG



Charts by Gary Bradski, 2005

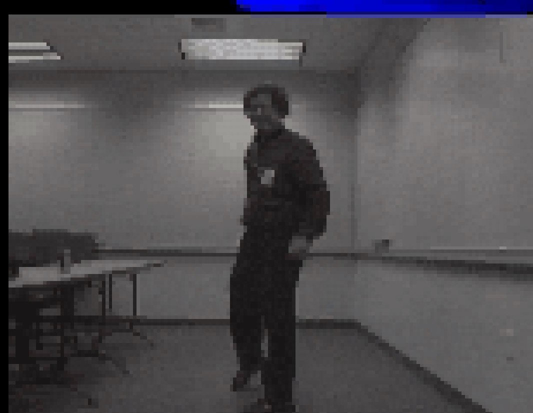
Segmentation, Motion Tracking and Gesture Recognition

Motion Segmentation



Motion Segmentation

Pose Recognition



Gesture Recognition

Screen shots by Gary Bradski, 2005

New Optical Flow Algorithms

```
// opencv/samples/c/lkdemo.c
int main(…){
…
CvCapture* capture = <…> ?
     cvCaptureFromCAM(camera_id) :
     cvCaptureFromFile(path);
if( !capture ) return -1;
for(;;) {
   IplImage* frame=cvQueryFrame(capture);
   if(!frame) break;
   // … copy and process image
cvCalcOpticalFlowPyrLK( …)
   cvShowImage( "LkDemo", result );
   c=cvWaitKey(30); // run at ~20-30fps speed
   if(c >= 0) {
       // process key
 }}
cvReleaseCapture(&capture
```

lkdemo.c, 190 lines

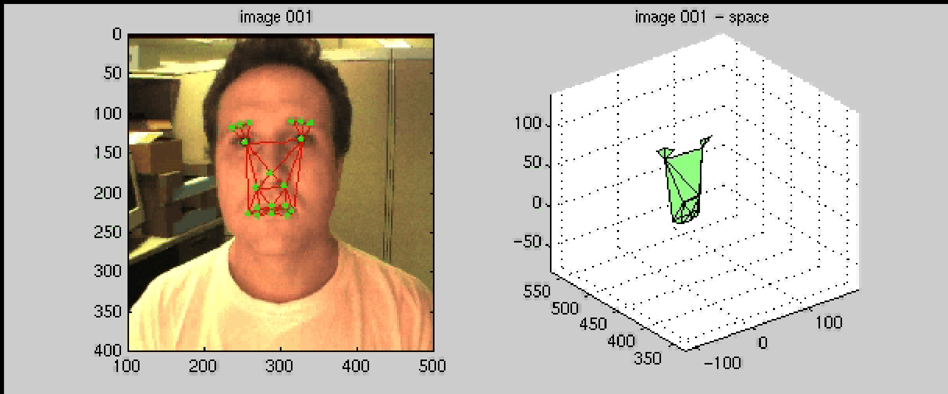
(needs camera to run)

$$I(x+dx, y+dy, t+dt) = I(x, y, t);$$

$$-\partial I / \partial t = \partial I / \partial x \cdot (dx / dt) + \partial I / \partial y \cdot (dy / dt);$$

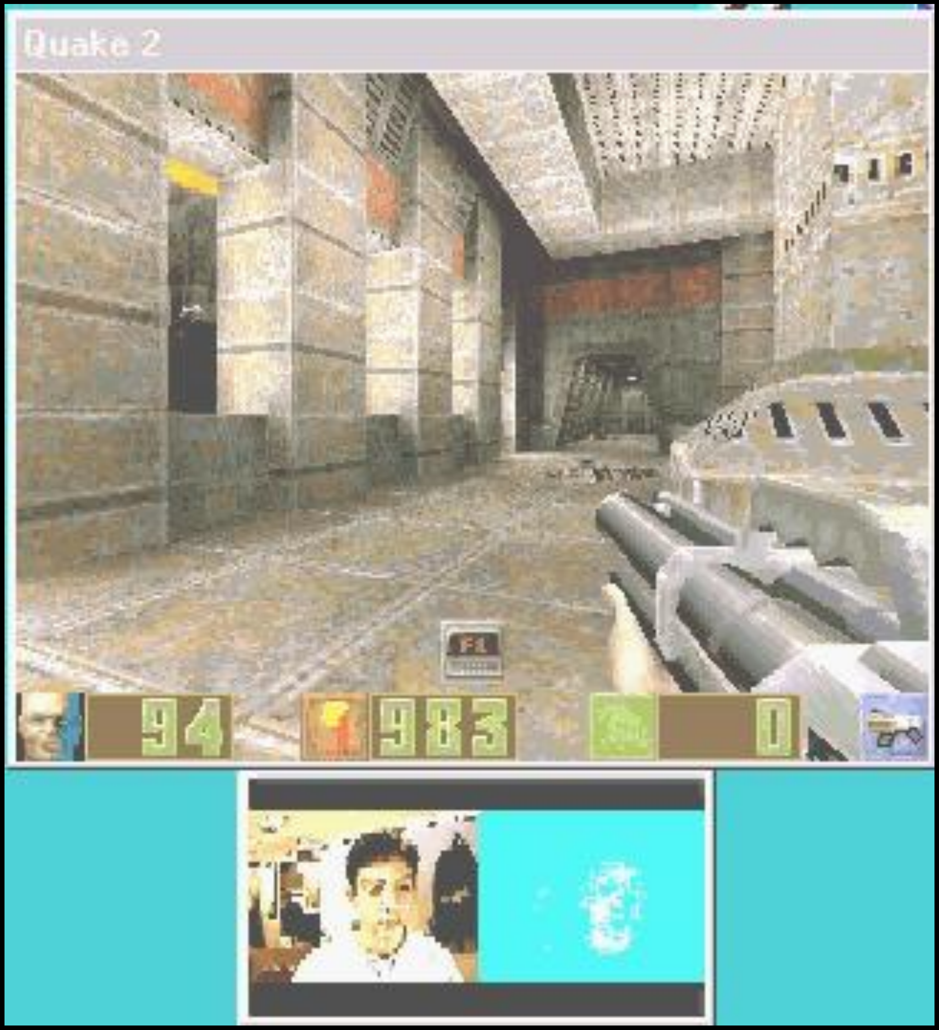
$$G \cdot \partial X = b,$$

$$\partial X = (\partial x, \partial y),\ G = \sum \begin{bmatrix} I_x^2, & I_x I_y \\ I_x I_y, & I_y^2 \end{bmatrix},\ b = \sum I_t \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$



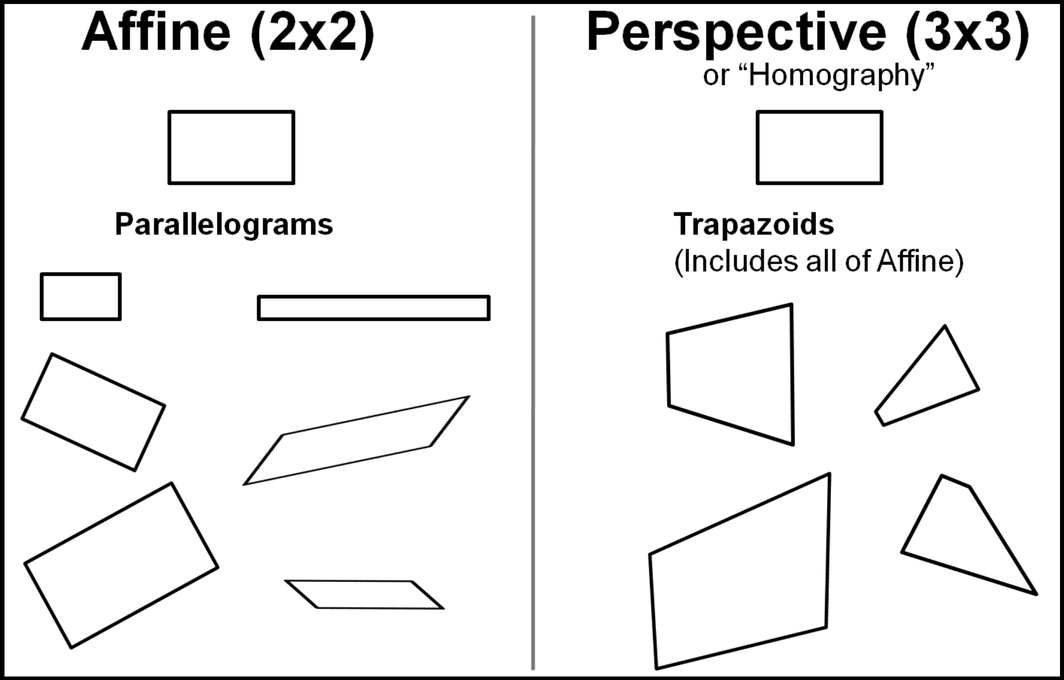
Tracking with CAMSHIFT

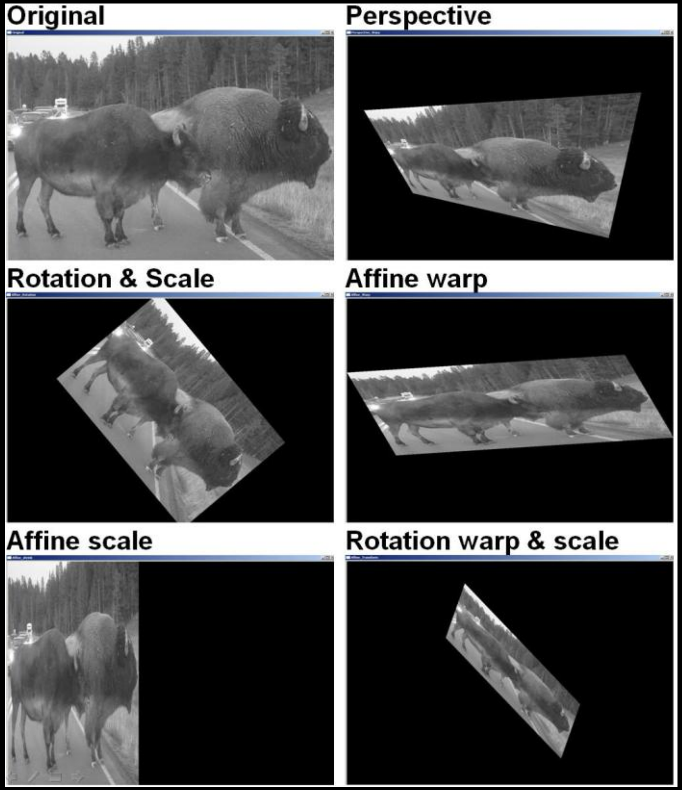
- Control game with head



Screen shots by Gary Bradski, 2005

Projections

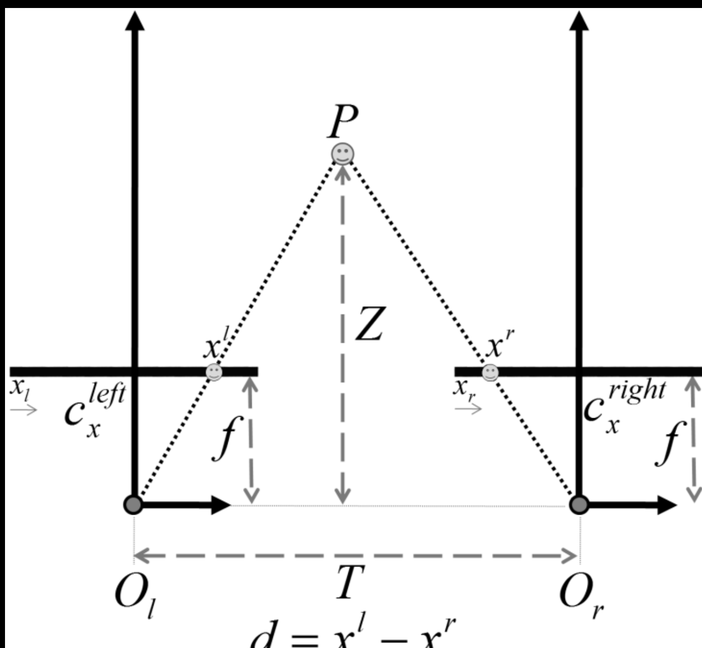




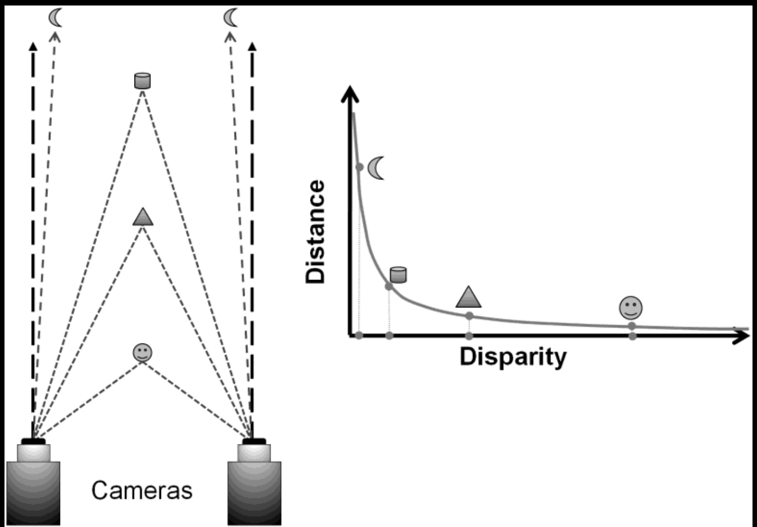
Screen shots by Gary Bradski, 2005

Stereo … Depth from Triangulation

- Involved topic, here we will just skim the basic geometry.
- Imagine two perfectly aligned image planes:

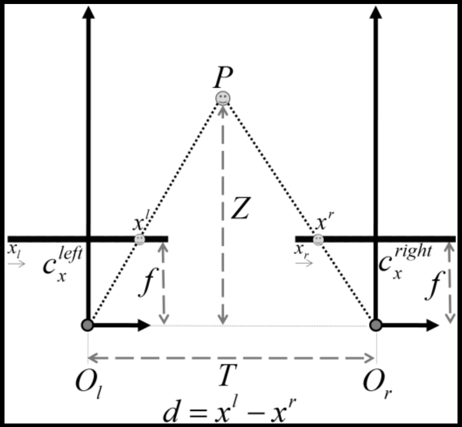


Depth "Z" and disparity "d" are inversly related:



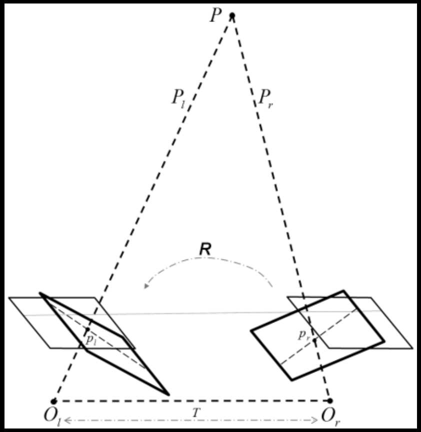
Stereo

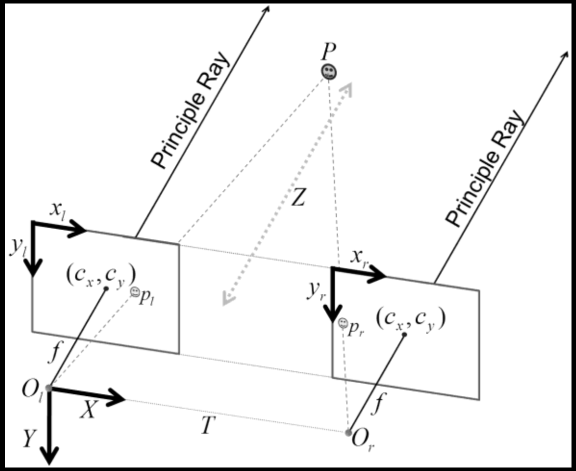
- In aligned stereo, depth is from similar triangles:



$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^l - x^r}$$

- Problem: Cameras are almost impossible to align
- Solution: Mathematically align them:



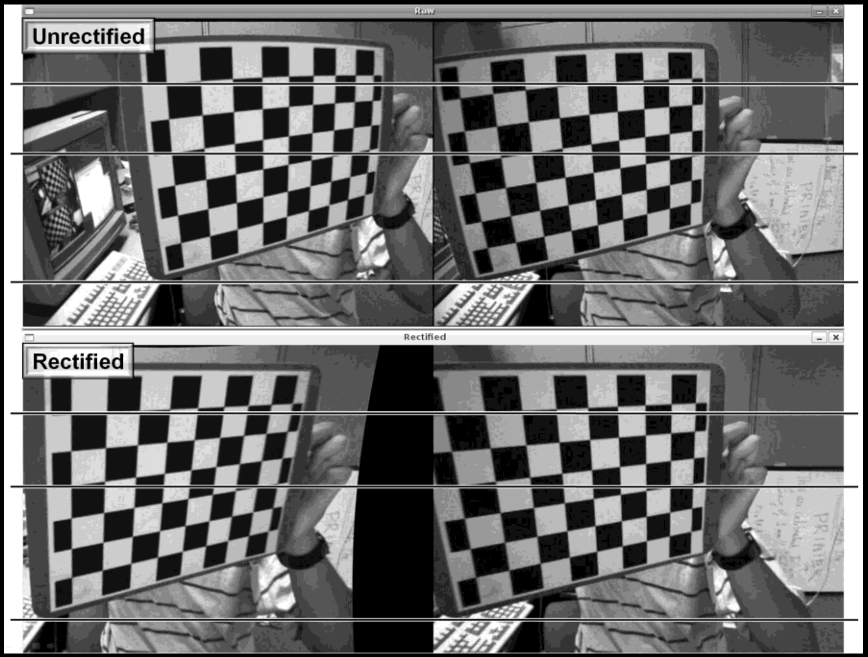


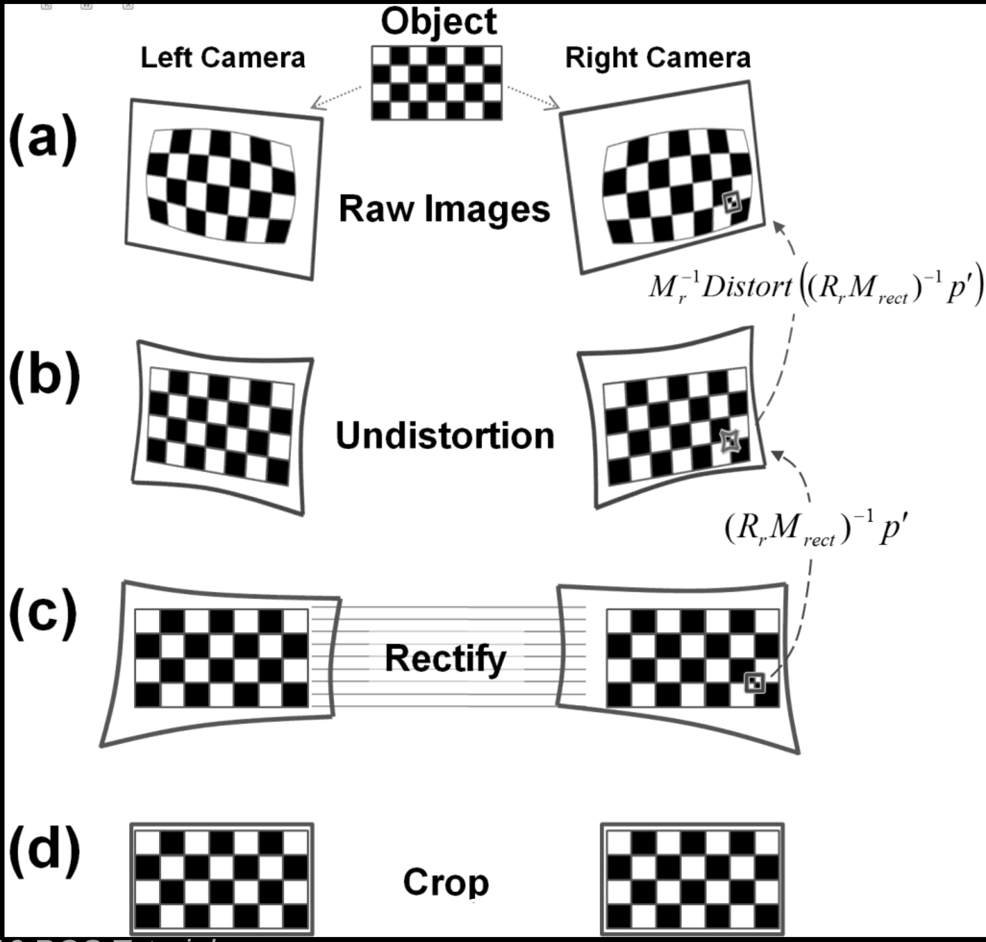
All: Gary Bradski and Adrian Kaehler: Learning OpenCV

Stereo Rectification

- Algorithm steps are shown at right:
- Goal:
  – Each row of the image contains the same world points
  – "Epipolar constraint"

Result: Epipolar alignment of features:





All: Gary Bradski and Adrian Kaehler: Learning OpenCV

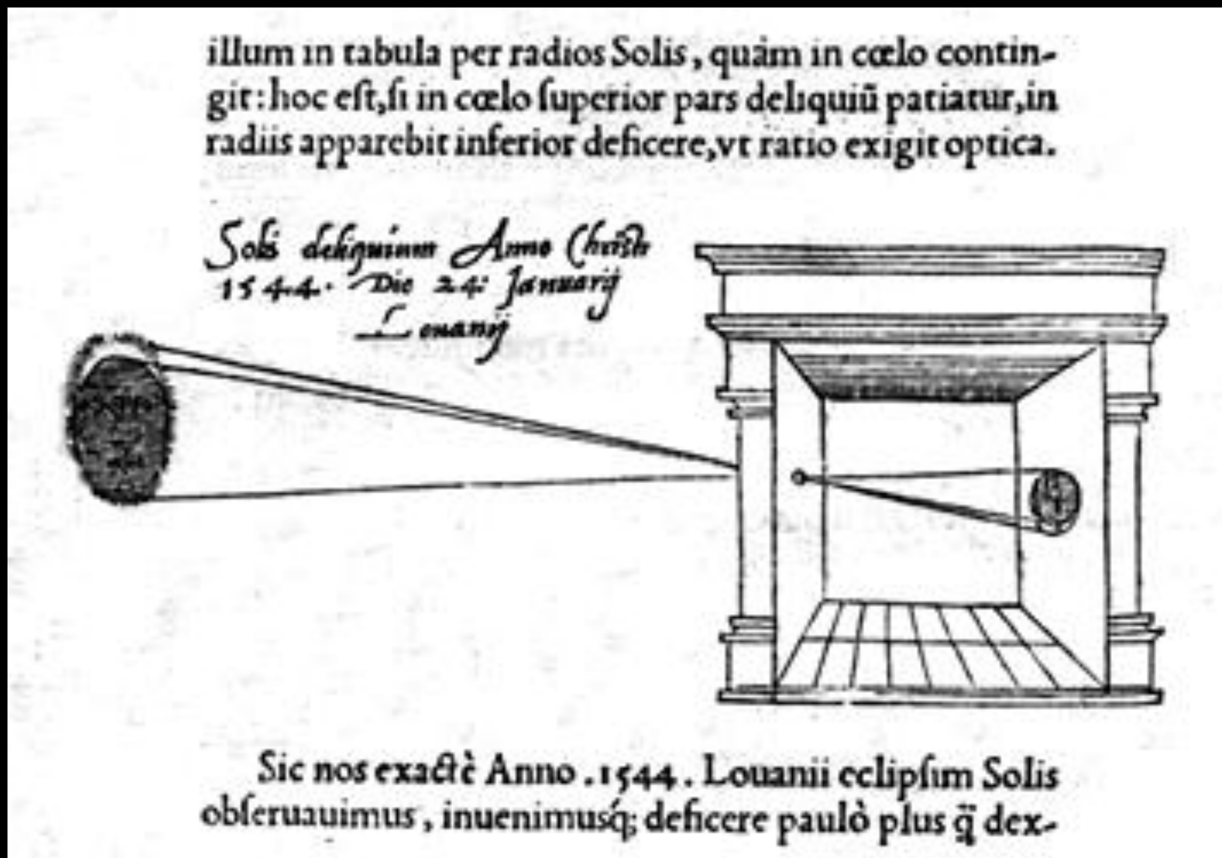
Summary

- Wide ranging functionality in
  - Matrix funtions
  - Image processing
  - Feature extraction
  - Segmentation
  - Motion
  - Stereo
  - Object recognition

Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

Most basic camera model: Pinhole Camera

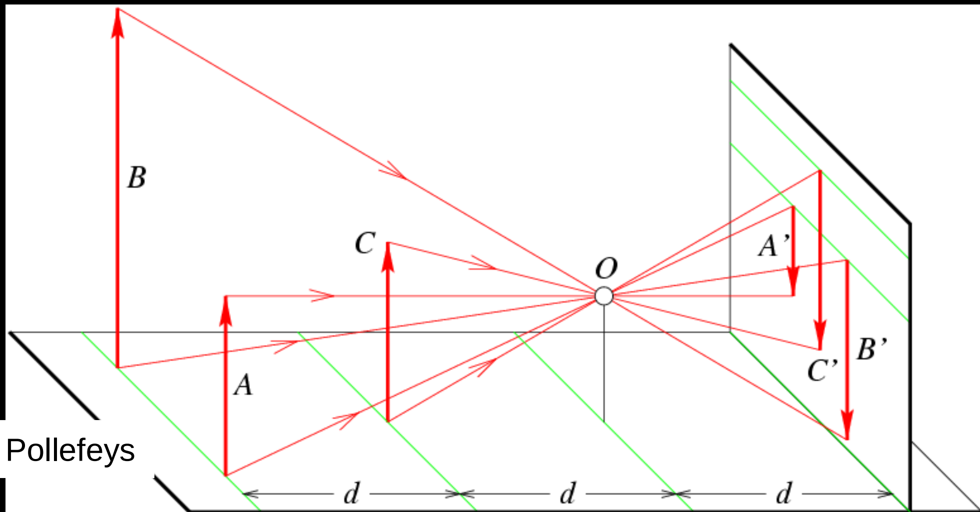

illum in tabula per radios Solis, quàm in cœlo contingit: hoc est, si in cœlo superior pars deliquiũ patiatur, in radiis apparebit inferior deficere, vt ratio exigit optica.

Sic nos exactè Anno .1544. Louanii eclipsim Solis obseruauimus, inuenimusq; deficere paulò plus q̃ dex-

-- Brunelleschi, XVth Century

Marc Pollefeys comp256, Lect 2

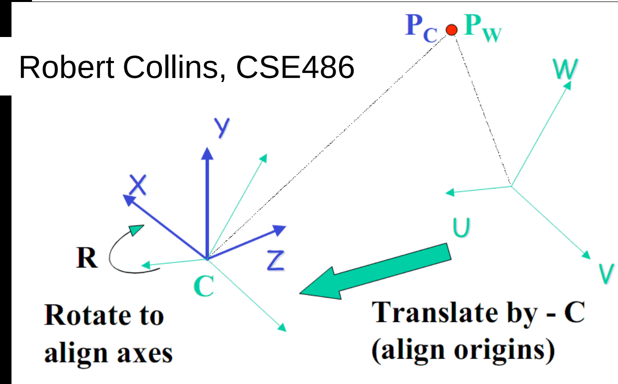
Perspective Projection



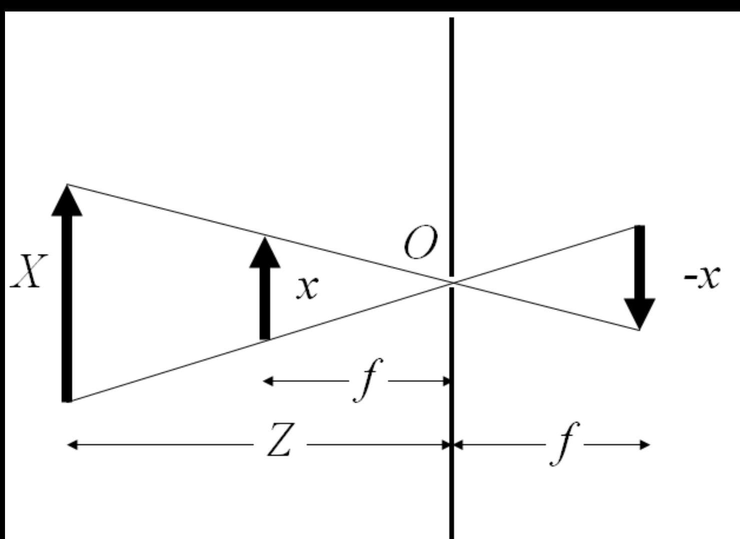
Marc Pollefeys

World to Camera Coordinates

Robert Collins, CSE486



$$P_C = R\,(P_W - C) = R\,P_W + T$$



$$x = -X\frac{f}{Z}$$

Perspective Matrix Equation

(camera coords Pt in world to pt on image)

$$x = f\frac{X}{Z} \qquad y = f\frac{X}{Z}$$

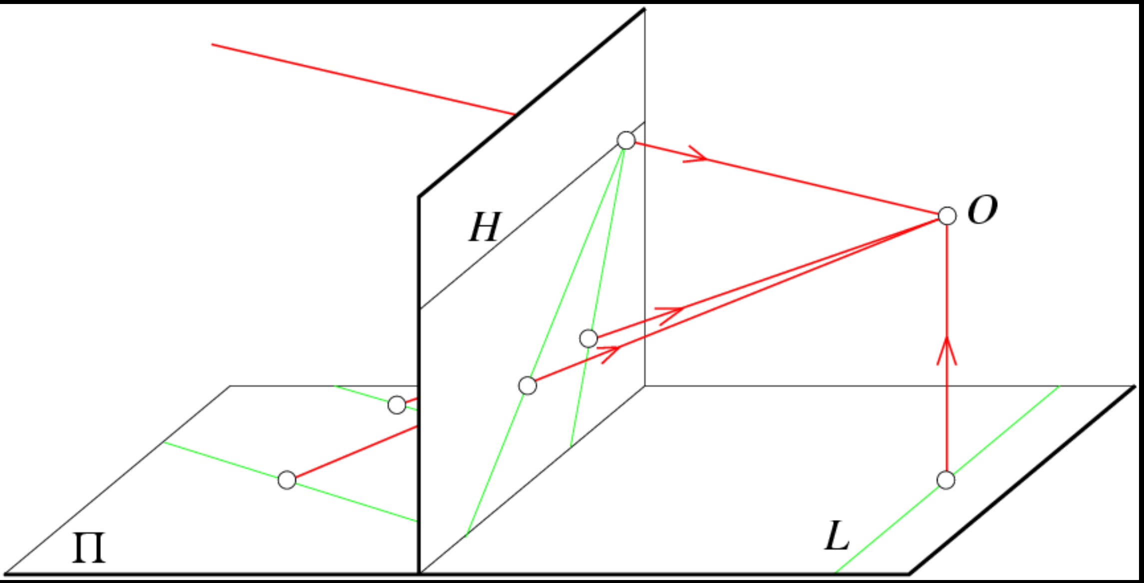
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$p = M_{\text{int}} P_C$$

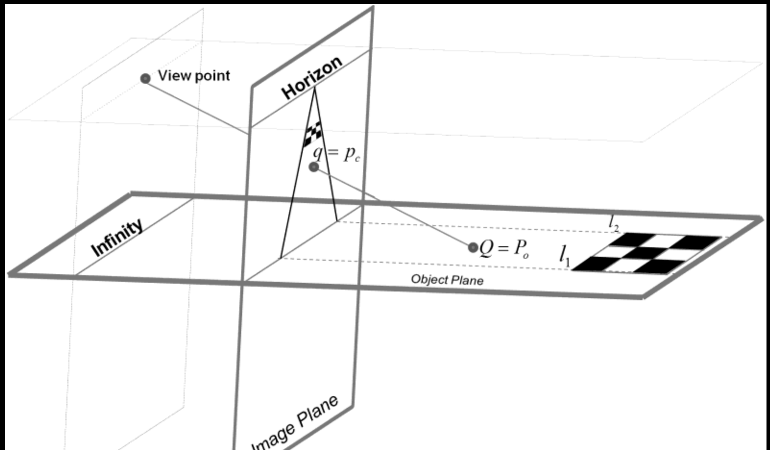
A “similar triangle’s” approach to vision.

Consequences: Parallel lines meet

- There exist vanishing points

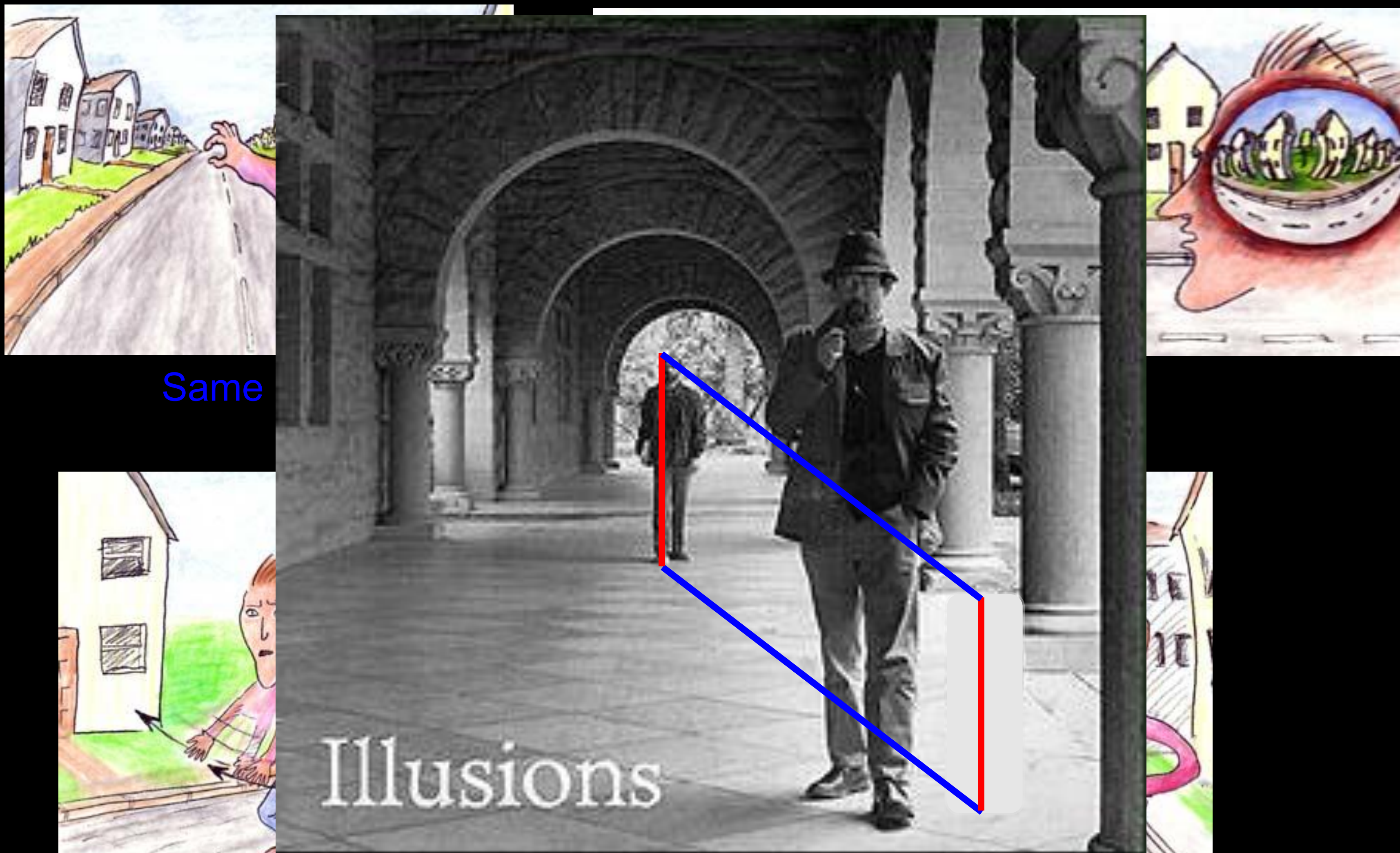


Marc Pollefeys



Gary Bradski and Adrian Kaehler: Learning OpenCV

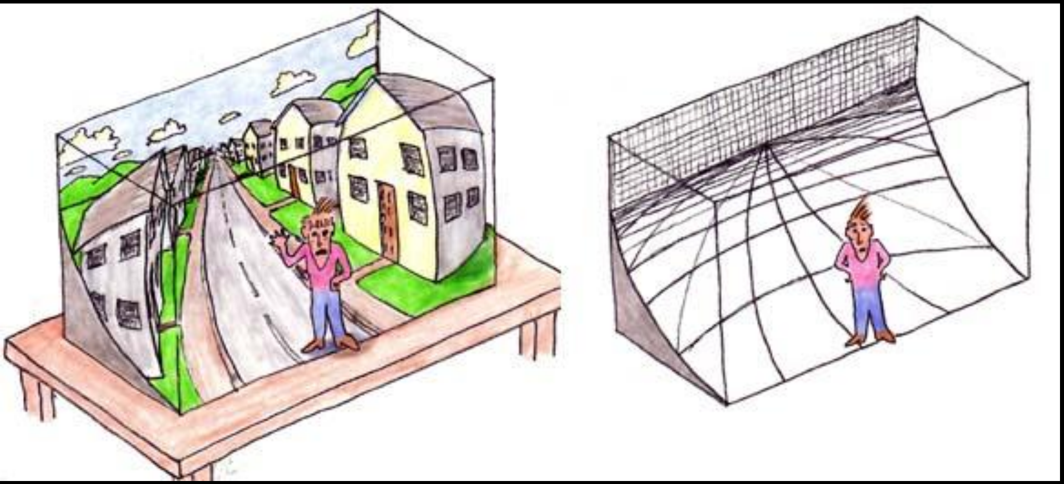
Implications For Perception*

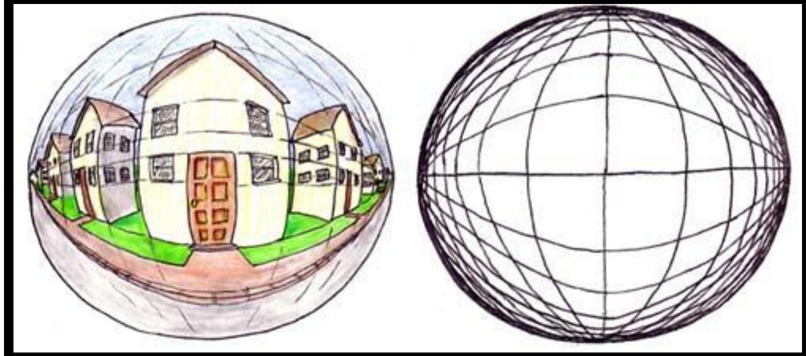


Parallel lines meet at a point…

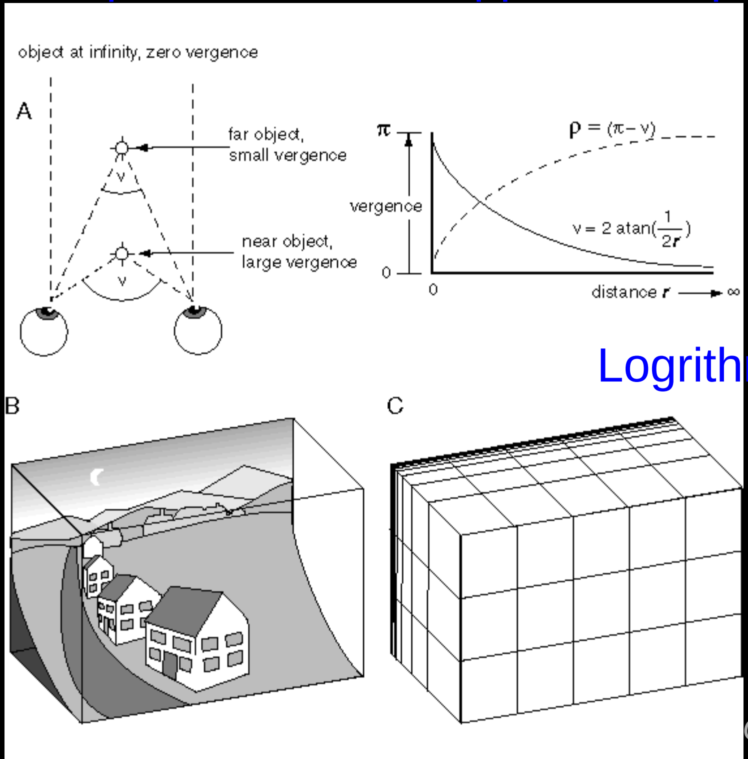
* A Cartoon Epistemology: http://cns-alumni.bu.edu/~slehar/cartoonepist/cartoonepist.html

Biological Implications For Perception:

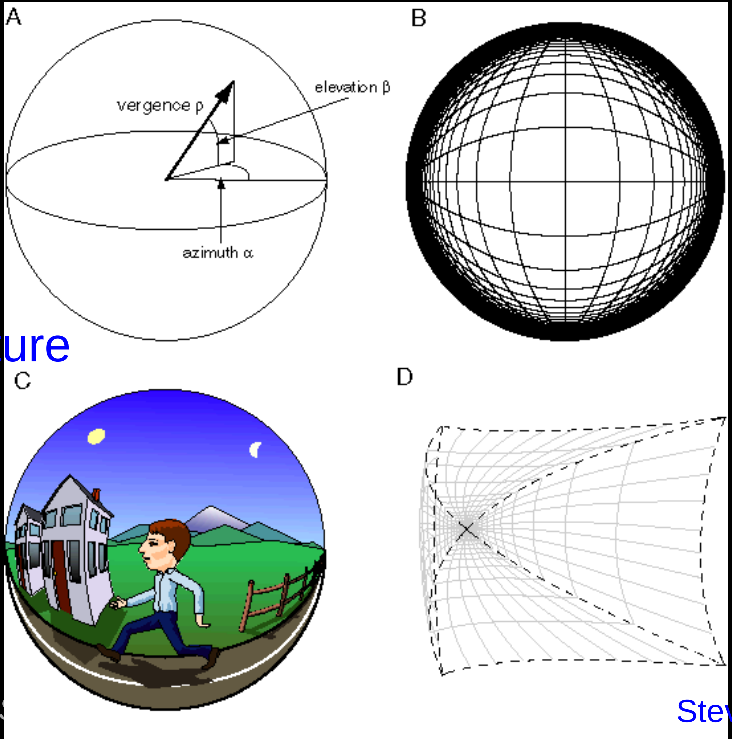




Perception must be mapped to a space variant grid



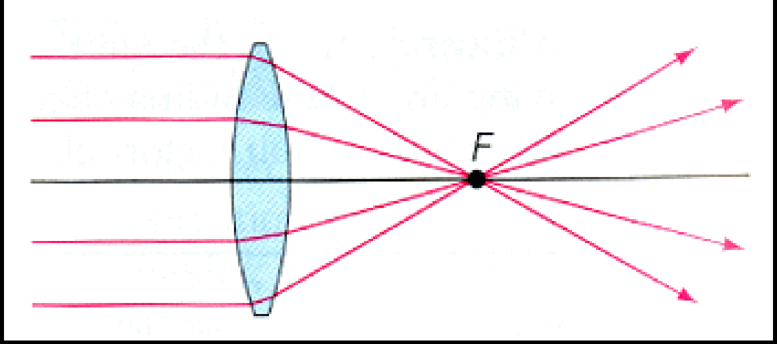
Logrithmic in nature



Steve Lehar

Pinhole Cameras are "Ideal" but…

- PROBLEM:
  - Pinhole cameras cannot gather enough light for practical applications.
- SOLUTION:
  - Use a lense to focus lots of light into a small area
- TRADEOFF:
  - Lense distortion



From: http://www.physics.uiowa.edu/~umallik/adventure/geo-optics/lightnw.htm

Aberrations

2 types :

1. geometrical

2. chromatic

geometrical : small for paraxial rays

study through 3rd order optics $\sin(\theta) \approx \theta - \frac{\theta^3}{6}$

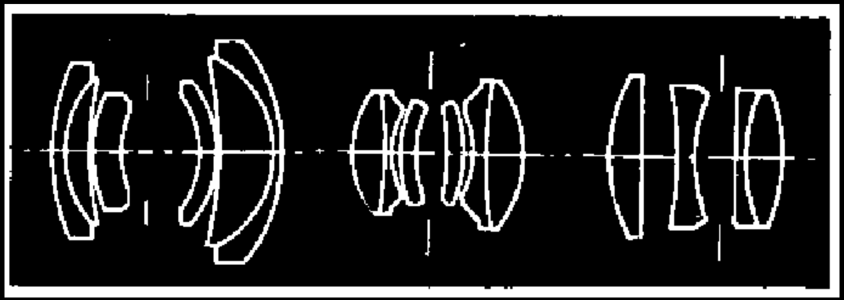
chromatic : refractive index function of wavelength

Marc Pollefeys

Geometrical aberrations

- spherical aberration
- astigmatism
- distortion
- coma

Aberrations are reduced by combining lenses
-- This is why a good lens costs to much.



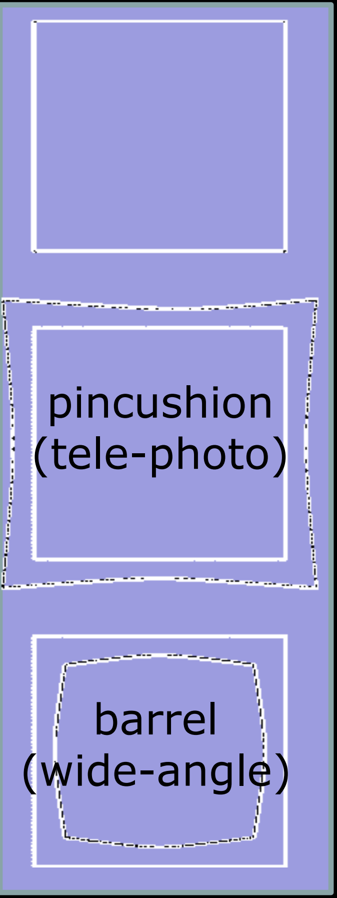
Marc Pollefeys

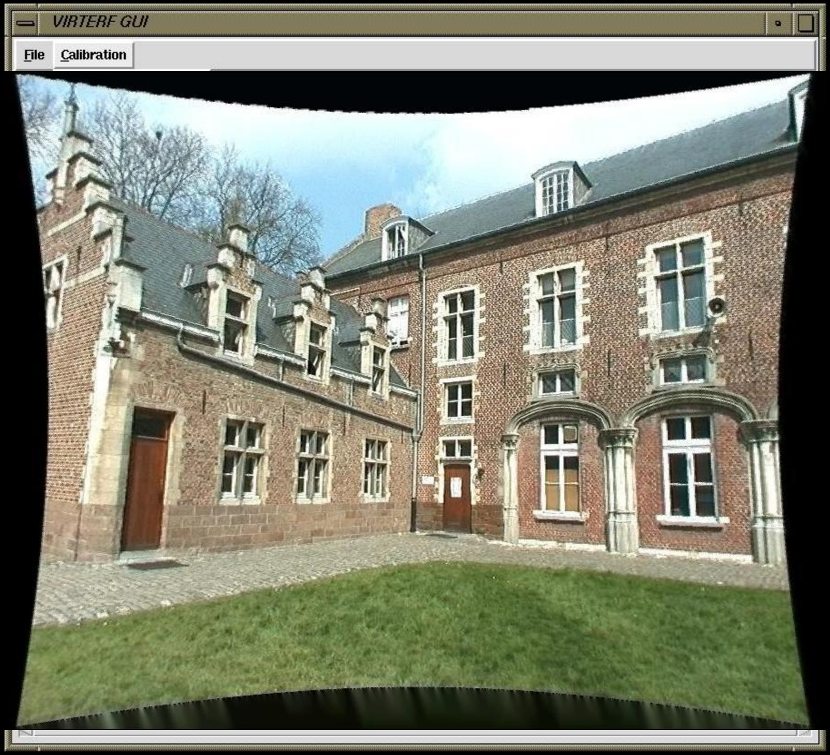
Geometric Distortion

(The major source of distortions)

RADIAL DISTORTION:

magnification/focal length different for different angles of inclination

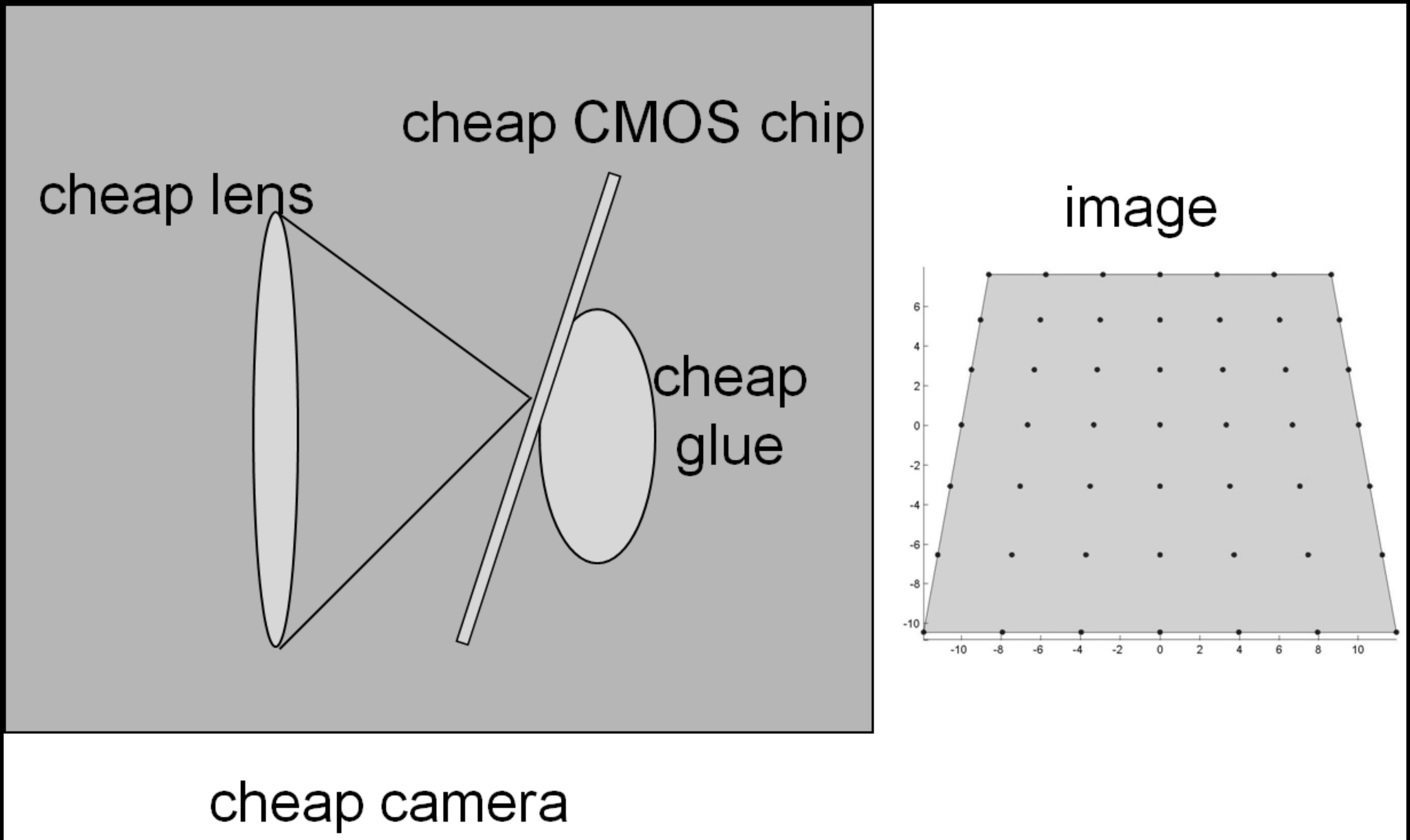




Can be corrected! (if parameters are know)

Marc Pollefeys

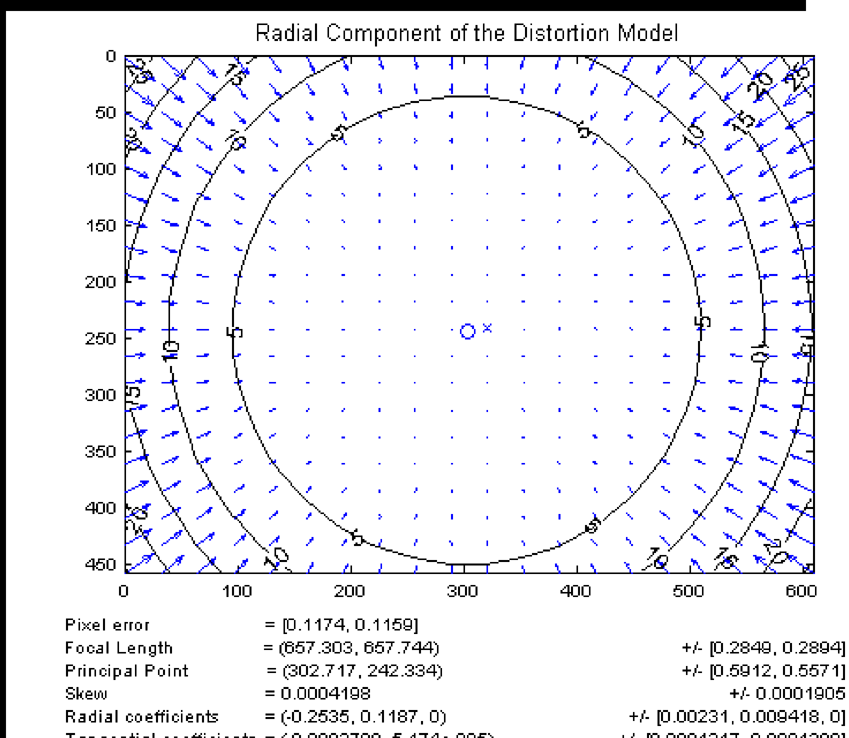
TANGENTIAL DISTORTION:

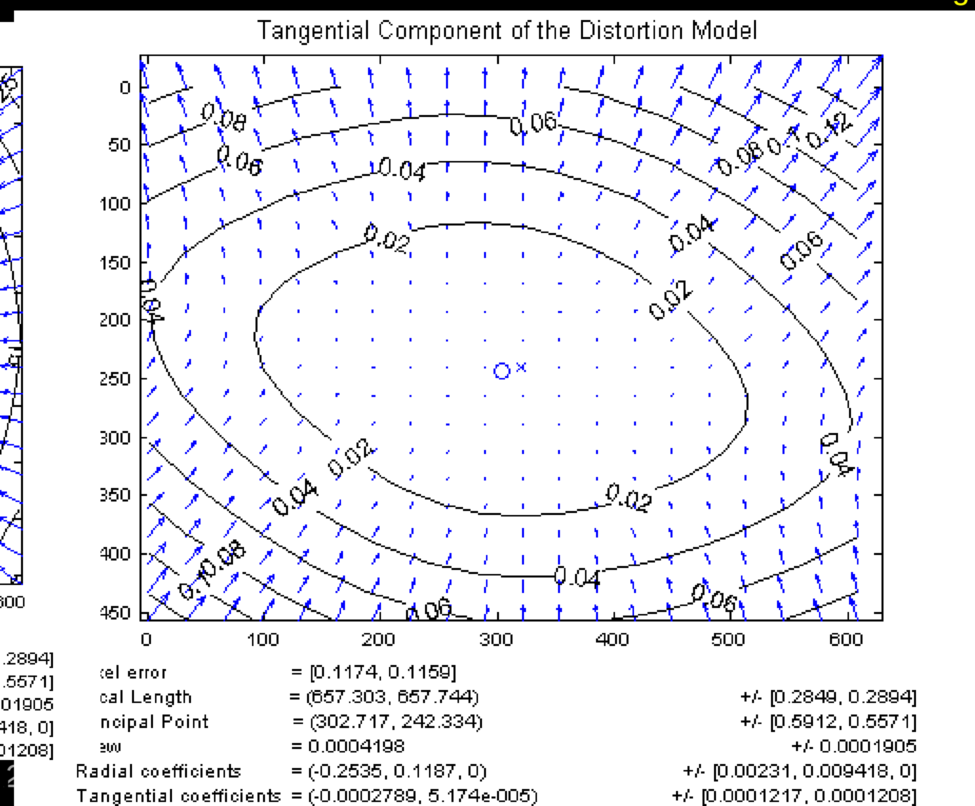


Distortion Field

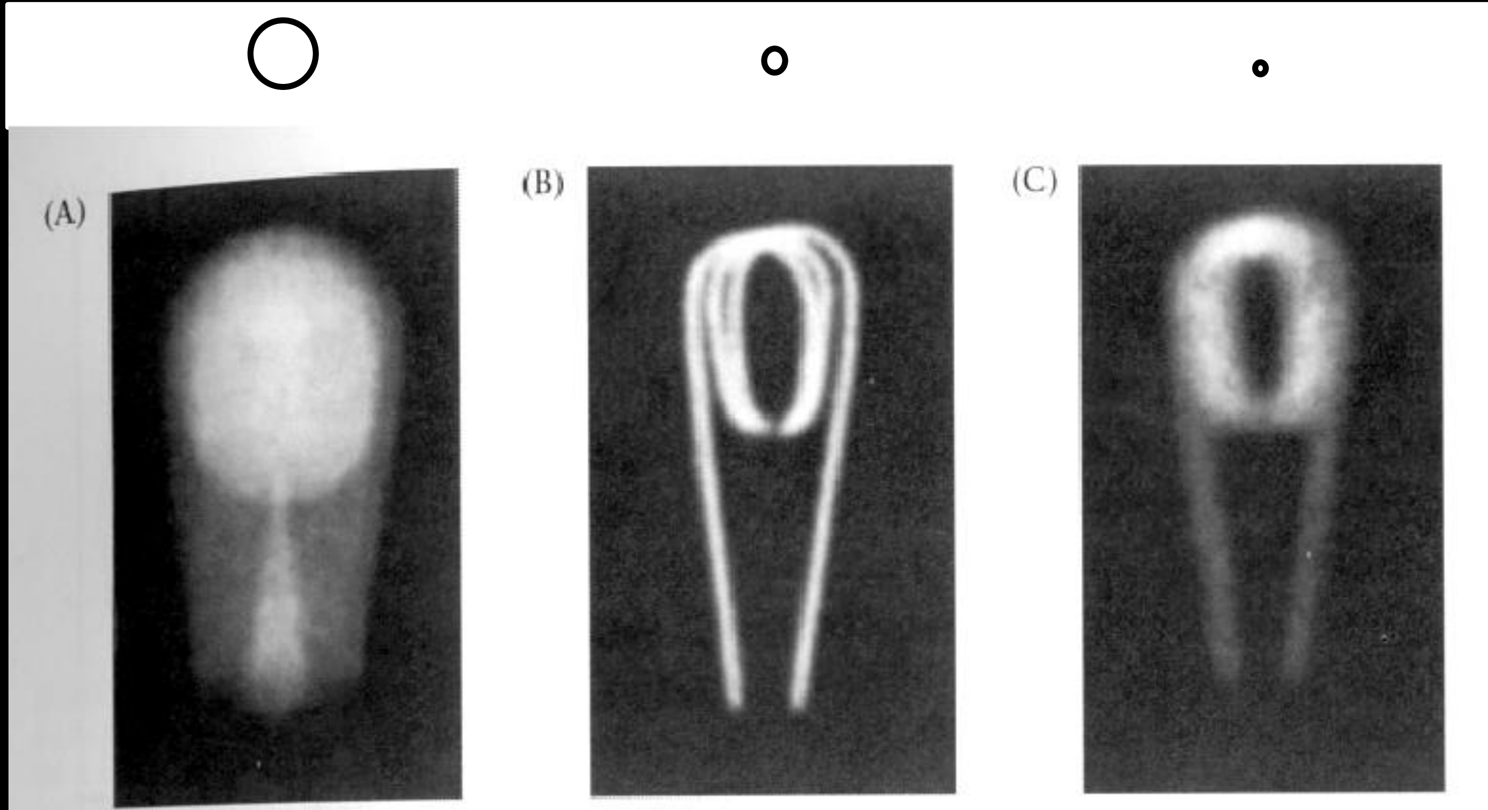
- Points from an object in the world should project to pinhole perspective locations.
- Charts of errors from “Ideal”:

Jean-Yves Bouguet





But, Pinhole Cameras also Have Problems:



2.18 DIFFRACTION LIMITS THE QUALITY OF PINHOLE OPTICS. These three images of a bulb filament were made using pinholes with decreasing size. (A) When the pinhole is relatively large, the image rays are not properly converged, and the image is blurred. (B) Reducing the size of the pinhole improves the focus. (C) Reducing the size of the pinhole further worsens the focus, due to diffraction. From Ruechardt, 1958.

Summary

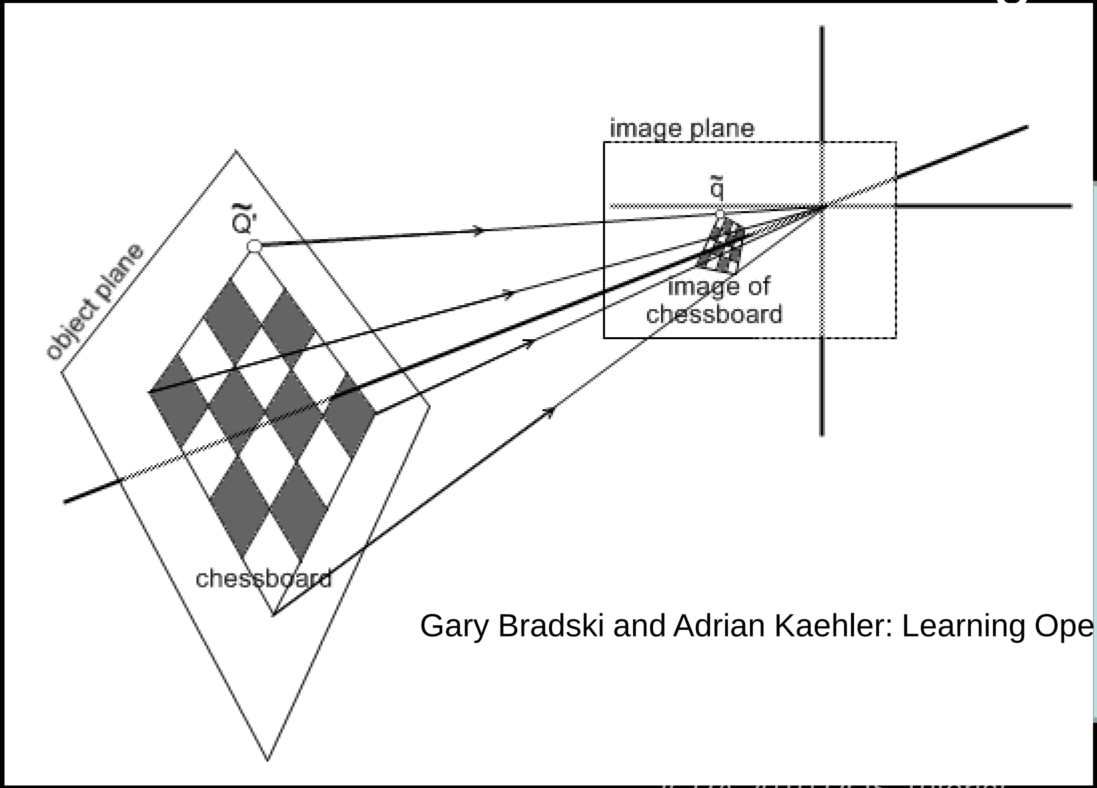
- Camera Model
  - Complications necessary to collect more light
- Simple camera, the pinhole, allows image formation
- Consequence is perspective mapping
- Pinholes don’t allow enough light
- Use lenses instead, but suffer from distortions

Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

Homography

- Maps one plane to another
  - In our case: A plane in the world to the camera plane
  - Great notes on this: Robert Collins CSE486
    - http://www.cse.psu.edu/~rcollins/CSE486/lecture16.pdf
  - Derivation details: Learning OpenCV 384-387



Gary Bradski and Adrian Kaehler: Learning OpenCV

Perspective Matrix Equation
(camera coords Pt in world to pt on image)

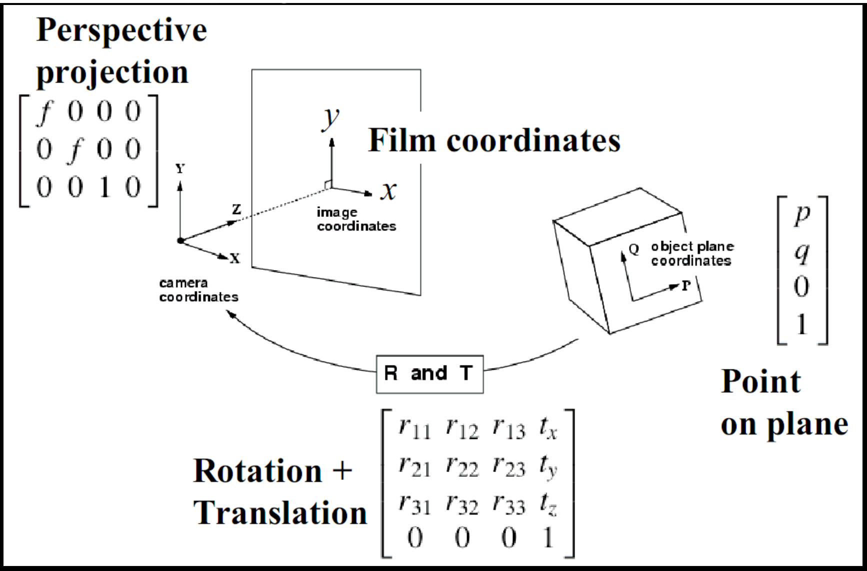
$$x = f\frac{X}{Z} \qquad y = f\frac{X}{Z}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$p = M_{\text{int}} P_C$$

Homography

Want: Mapping of Pt on Object to pt on image:



Robert Collins, CSE486

Projection Derivation (1):

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

Robert Collins, CSE486

Projection Derivation (2):

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} fr_{11} & fr_{12} & ft_x \\ fr_{21} & fr_{22} & ft_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

Homography H (planar projective transformation)

Robert Collins, CSE486

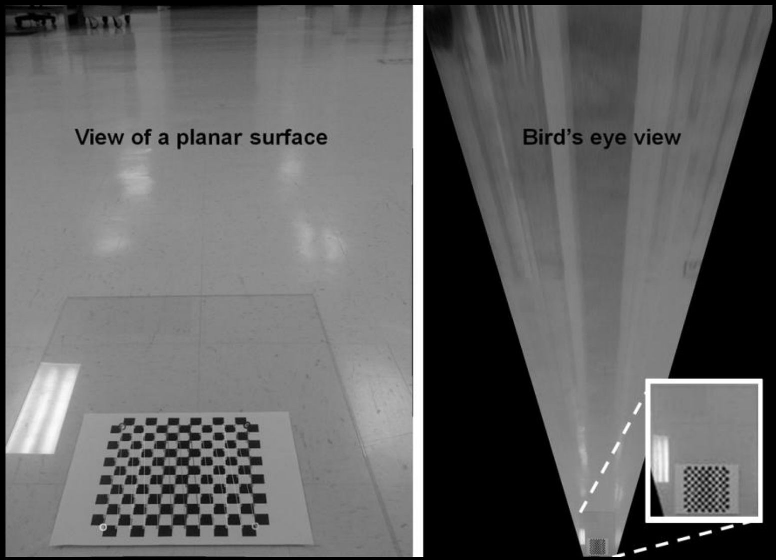
In reality, a plane 2x as big 2x as far away yields the same image. Thus t_z is arbitrary and is often fit to known coordinates by scaling. Thus H has only 8, not 9 parameters.

Homography

- We often use the chessboard detector to find 4 non-colinear points
  - (X,Y * 4 = 8 constraints)
  - To solve for the 8 homography parmeters.
- **Code:** Once again, OpenCV makes this easy
  - **findHomography(…)** or:
  - **getPerspectiveTransform(…)**

Homography Uses

- If you have a known planar object on the ground plane,
  - you can use it to map any other ground pt in the image to its (X,Y,Z) point on the ground



We used this in the DARPA Grand Challenge to map the image road segmentation to a bird's eye view obsticle map:

Parking a robot

```
getPerspectiveTransform(objPts,imgPts,H);   //This learns ground_pts->image_pts
invert(H,H_invt);                           //So we need to invert this to get img_pts->ground_pts
```

Summary

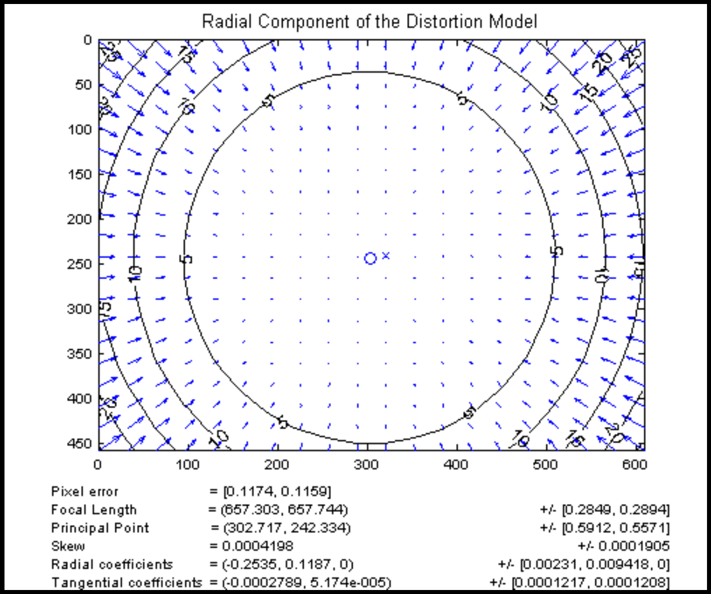
- Homography
  - Uses
- Given 4 image points corresponding to 4 non-colinear points on an external planar object
- We can compute a mapping from image points to points on the external plane
- Robots can use this to know where things are in an external ground plane.

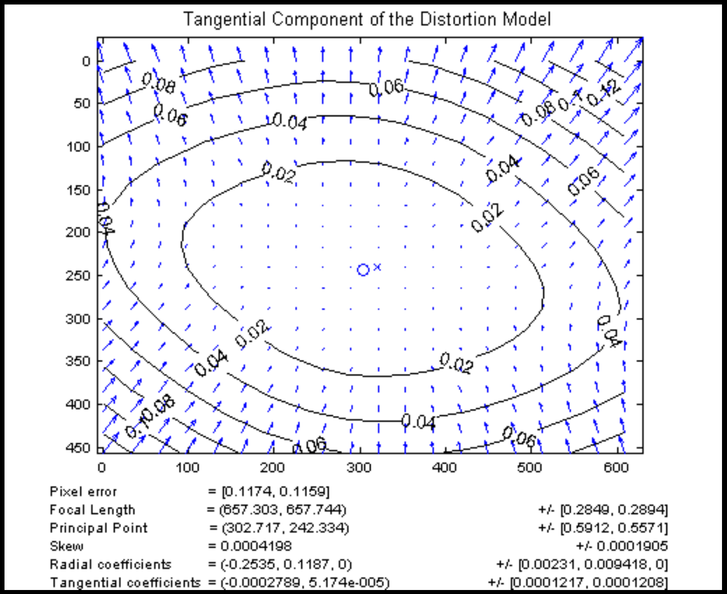
Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

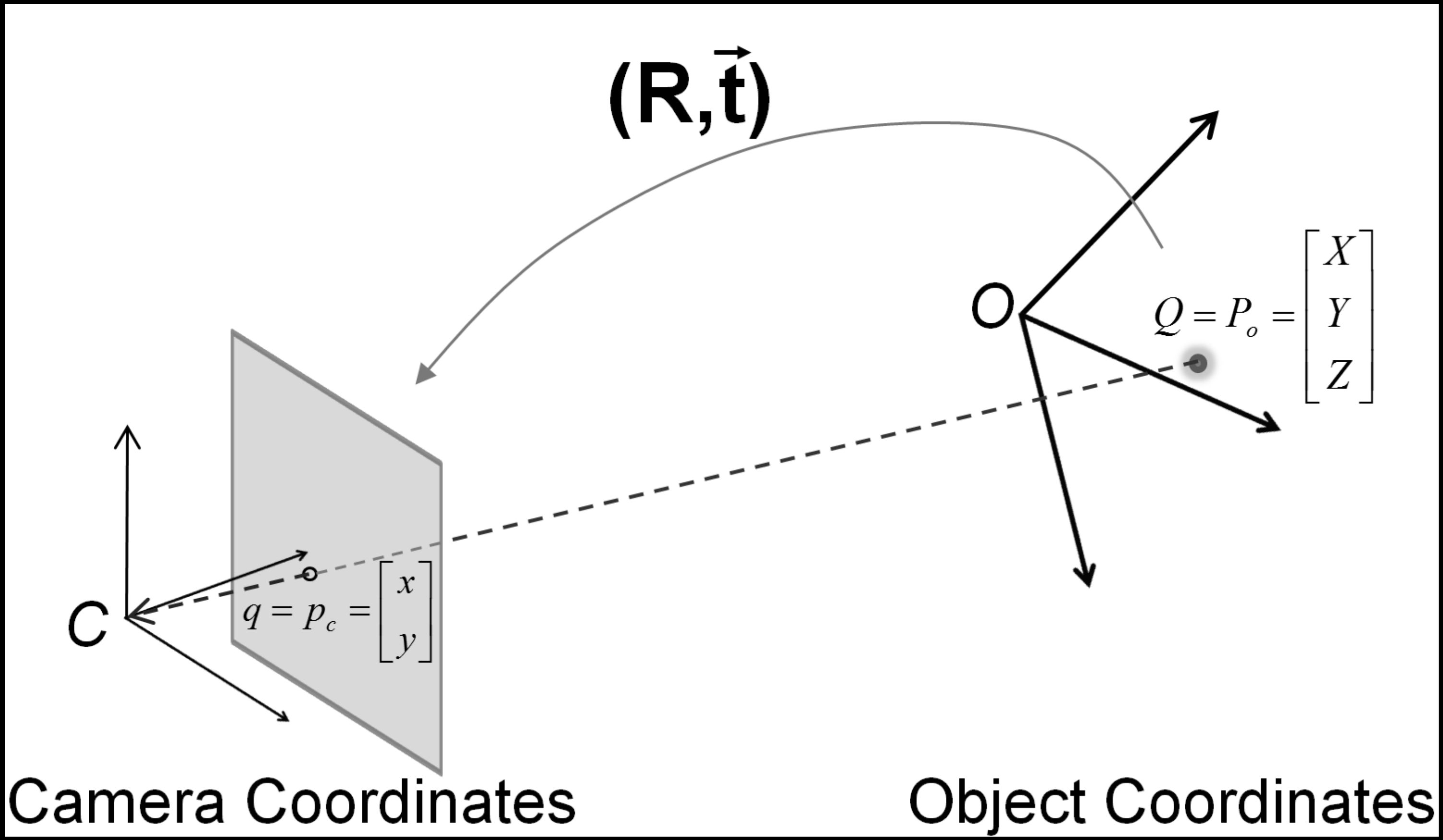
Calibration Intuition

- Points are distorted from their “ideal” positions in a pinhole (pure perspective) camera.
- Take images of an object with known points.
- Find out where those points are in the camera image.
- Compute a tranformation that maps the points to the correct “ideal” positions they should have.



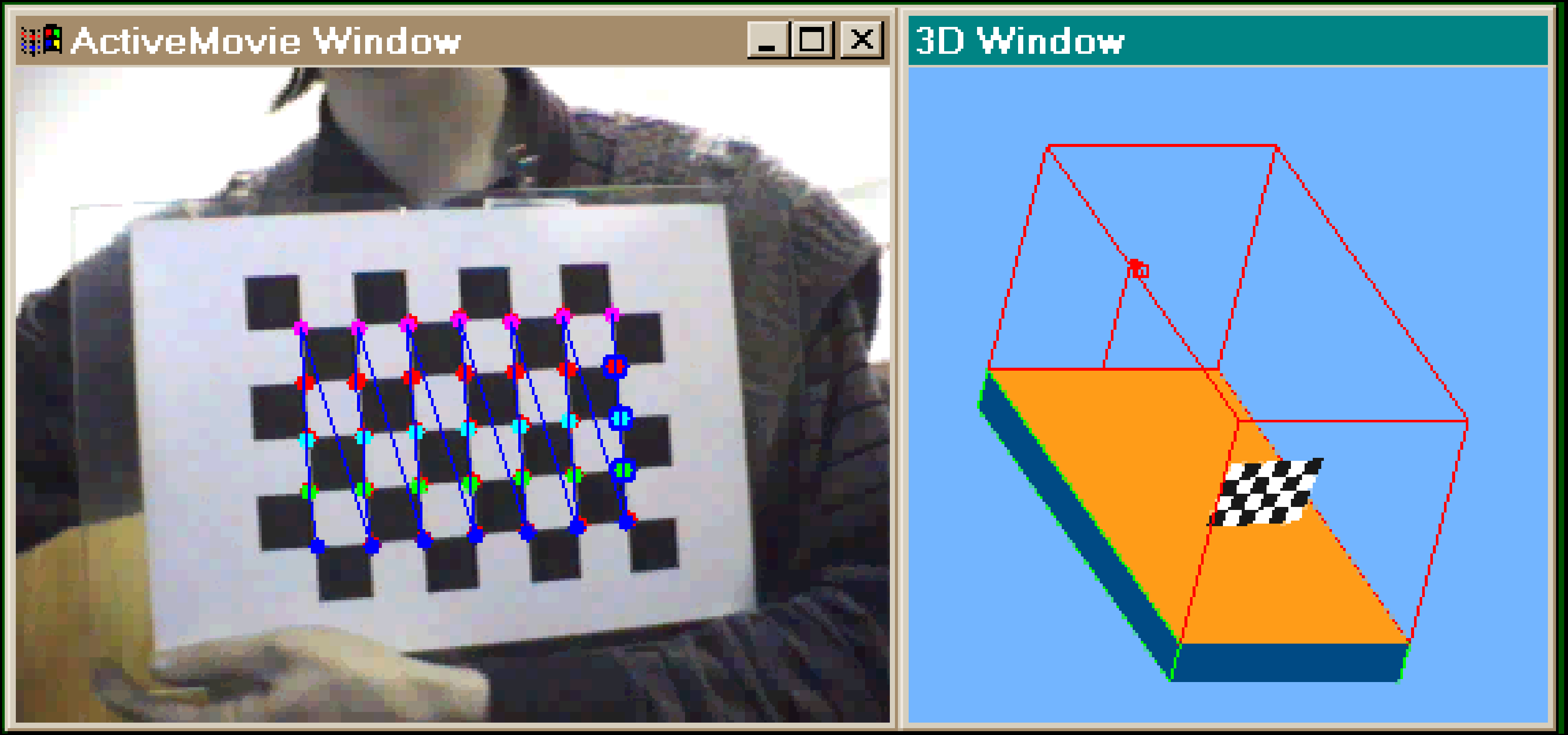


Relate Known Object to Camera Positions



Calibration Object

- A chessboard makes a good calibration pattern.
- We find its corners using the Harris corner detector.
- Calculate its homography and plot it.



Camera Intrinsic and Extrinsic Parameters

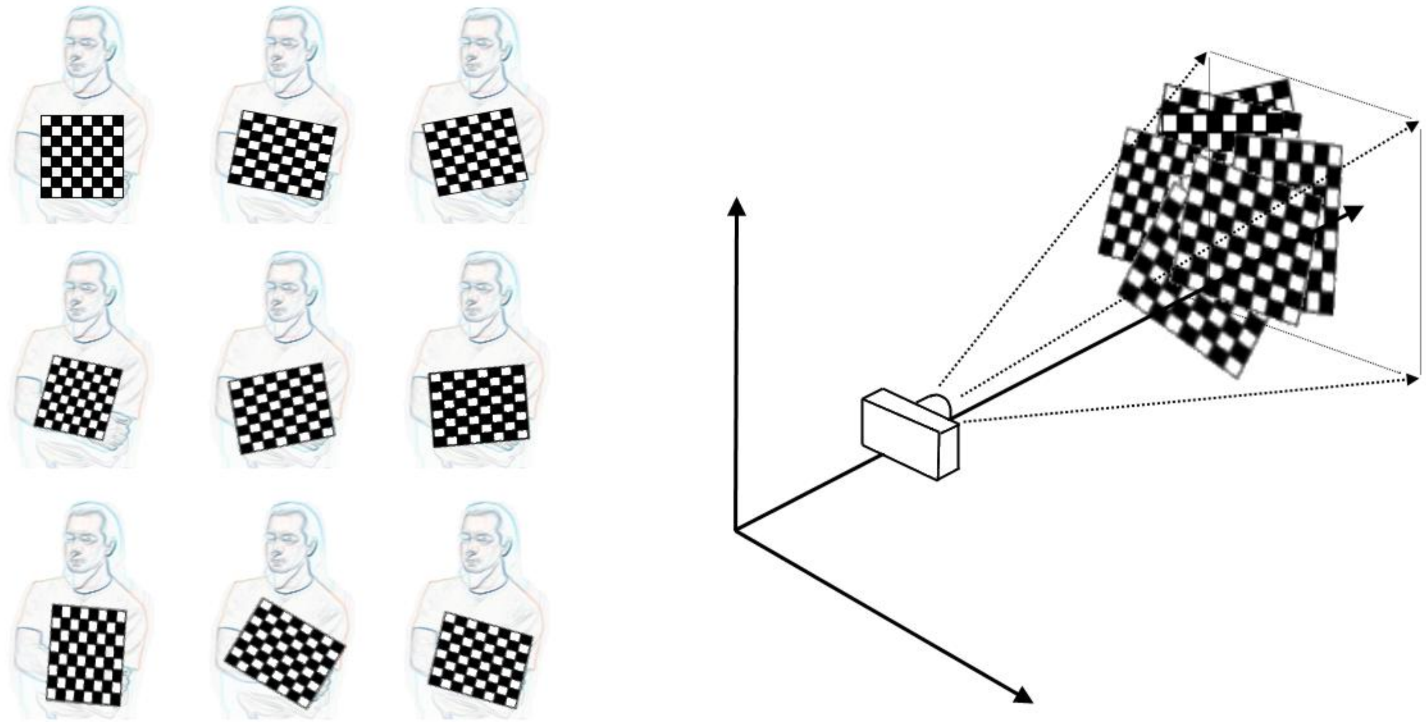
- Intrinsics:
  - Focal Length f
  - Pixel size s
  - Image center o_x, o_y
- Extrinsics:
  - Location and orientation of *k*-th calib. pattern:

$$\phi, \varphi, \psi, \quad T$$

Number of Images Needed

- K images of chessboads with N corners
- Have 4 intrinsic parameters (f,s,c(x,y)) and 6 extrinsic (x,y,z, pan, tilt, yaw).
- Solving then requires 2NK>6K+4
  - (N-3)K>=2
  - If N=5, K= 1
  - NO! Planar chessboard has only N=4 amount of information, so K >= 2.
  - In practice, we use many images to reduce noise.

Calibration Object Image Collection



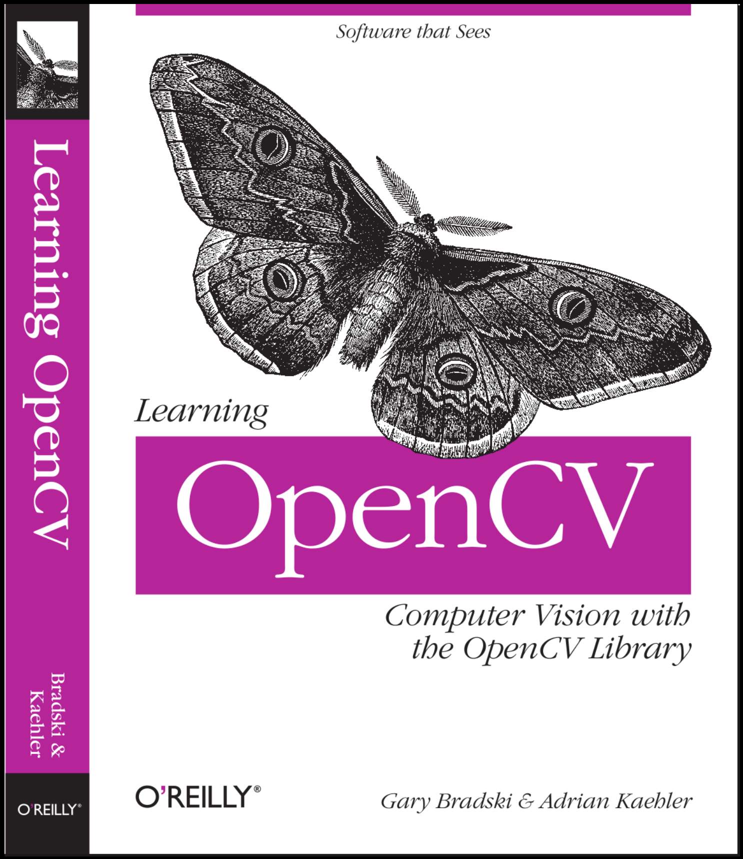
Gary Bradski and Adrian Kaehler: Learning OpenCV

Solve the Full Perspective Camera Model

$$\begin{pmatrix} x_{im} \\ y_{im} \end{pmatrix} = \left[\begin{pmatrix} -\dfrac{f}{s_x} \dfrac{\begin{pmatrix} \cos\phi & \sin\phi & 0 \end{pmatrix} \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} X^W \\ Y^W \\ Z^W \end{pmatrix} + T_X}{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} X^W \\ Y^W \\ Z^W \end{pmatrix} + T_z} + o_x \\ -\dfrac{f}{s_y} \dfrac{\begin{pmatrix} -\sin\phi & \cos\phi & 0 \end{pmatrix} \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} X^W \\ Y^W \\ Z^W \end{pmatrix} + T_Y}{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} X^W \\ Y^W \\ Z^W \end{pmatrix} + T_z} + o_y \end{pmatrix} \right]$$

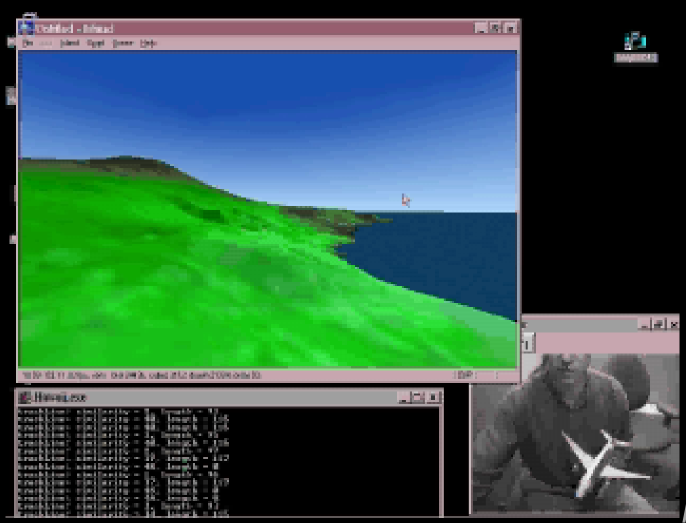
Code

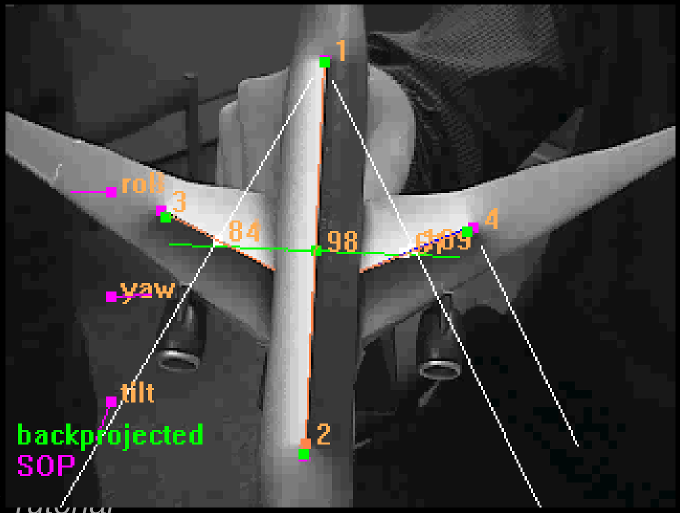
- Explained in detail in “Learning OpenCV”
- Calibration solved using OpenCV function:
  - **CalibrateCamera()**
  - OpenCV site:
  - http://opencv.willowgarage.com



Perspective n-Point Problem

- Many useful things follow from our calibration solution.
  - **Homography**:
    - Relating one plane to another (next section)
  - **PnP problem**:
    - If we can find known points on a known 3D object,
    - we can (we just did) find it's pose (it's orientation relative to the camera coordinate system).
    - **OpenCV CODE: solvePnP(…)**





Training Set Assist

- Find chessboard
- Relate another part of the scene to it

Summary

- Camera Calibration
  - How to compute
  - PnP problem solution
- Use a known object
- Find its “raw” projection to the camera plane
- Compute a mapping that moves the real location of features to the ideal locations.
- Same techniques useful for finding the pose of an object given known points (PnP Problem)

Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

Why is Vision Hard?

The difference between seeing and perception.

We perceive this:

Gary Bradski, 2005

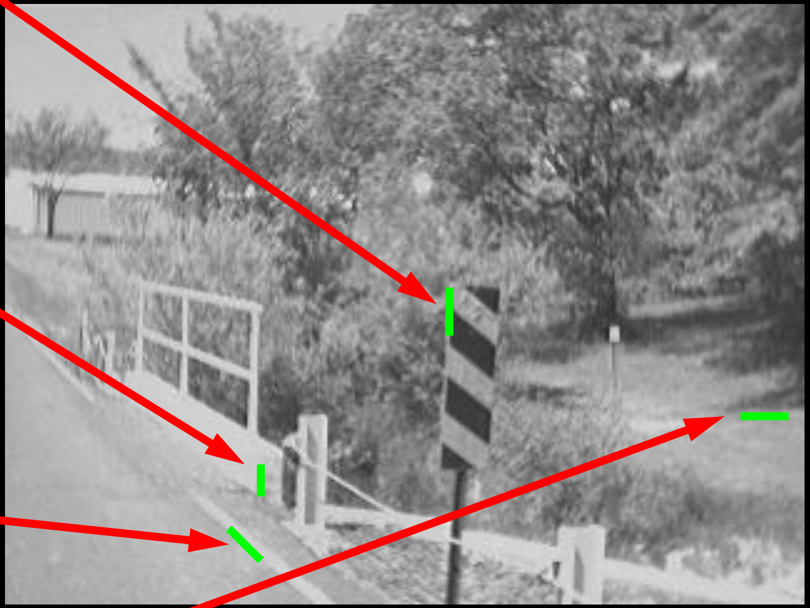
What to do? Search for features
Maybe we should try edges ….

But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

But, What's an Edge?

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., shadow)



Slide credit: Christopher Rasmussen

Gradient Features: Sobel

Typically gradients are found by convolving with a deriviatve operator

$$f[m,n] = I \otimes g = \sum_{k,l} I[m-k, n-l]\, g[k,l] \qquad \sum_{k,l} g[k,l] = 1$$

The Sobel derivative (approximates a Gaussian in one dimension, a difference operator in the other) is a popular choice:

$S_1 =$

-1	-2	-1
0	0	0
1	2	1

$S_2 =$

-1	0	1
-2	0	2
-1	0	1

$$\text{Edge Magnitude} = \sqrt{S_1^2 + S_1^2}$$

$$\text{Edge Direction} = \tan^{-1}\left(\frac{S_1}{S_2}\right)$$

OpenCV Code: sobel(…), but we prefer the **scharr(…)** operator since it is just as fast but more accurate at angles near +/- 45 degrees.

Binary Gradient Grids

Many features (notably Lowe's SIFT) start from collections of gradients

$$m = \sqrt{(L_{x+1,y} - L_{x-1,y})^2 + (L_{x,y+1} - L_{x,y-1})^2}$$

$$\theta = \tan^{-1}((L_{x,y+1} - L_{x,y-1})/(L_{x+1,y} - L_{x-1,y}))$$

And process the gradients into summary vectors:

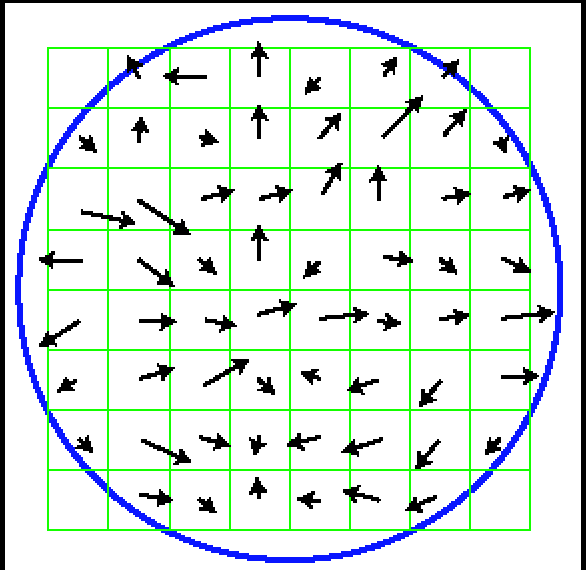


Image gradients

I will demo direct use of spatial layouts of binarized gradient grids:

Summary

- Vision is hard because of the 2D-3D ambiguity
  - Perception in contrast to seeing
- Therefor, use statistical regularity
  - Expreeess the image in terms of features
- We focus on binary gradient grids

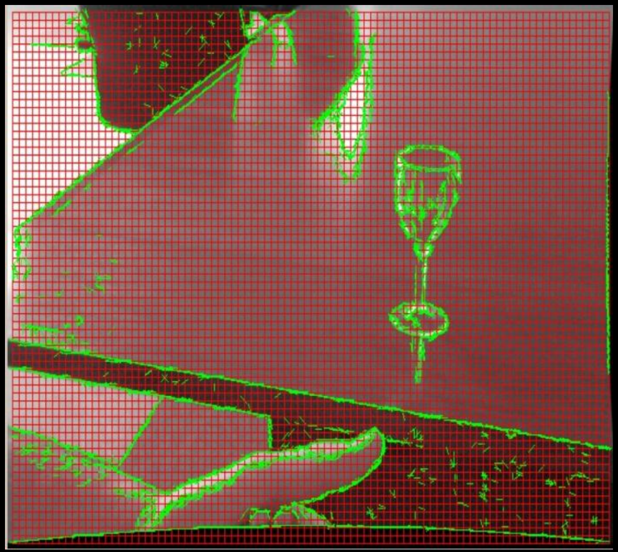
Outline

- Overview
- Quick Tour
- Pinhole Camera
- Homography
- Camera Calibration
- Gradient Features
- Demo
  - Object Recogntion Using Gradient Features
  - Node: Finding a Chessboard and its Pose

Binary Gradient Grid

We organize gradients in a way that allows extremely rapid search
This allows us to directly scale recognition with compute cycles

Raw



Gradient Grid

Recognized

Gary Bradski, 2010

Demo

Chessboard Service Node

```
# Get to the directory
roscd icra_ros_tutorials

# Start roscore and the checkerboard-detector-to-pose service
roslaunch launch/detectors.launch

# Play some data (or get from the robot)
while true; do rosbag play  checkerboard_box.bag; sleep 0.1; done

  # What nodes are up?
  rosnode list

  # What services are offered
  rosnode info /narrow_cb_detector
    #or
  rosservice list

  # What arguments do I need to enter for that service?
  rosservice args /narrow_get_checkerboard_pose

# Call the service
rosservice call narrow_get_checkerboard_pose 3 4 .108 .108
```

Demo

Q1: How do you call the wide stereo?

Q2: Are the results different? If so: Why?

Q3: If you change around the 3 and 4, what happens? Show why.

Summary

- We made use of what we learned:
  - Recognition using binary gradient grids
  - A node that finds a chessboard and returns its pose

Questions?

Photo: Gary Bradski