

# Sensor-enabled Intelligent Environments - Part 3 Vision



Nico Blodow, Mihai Dolha, Zoltan-Csaba Marton,  
Oscar Martinez Mozos, Dejan Pangercic@IAS

14. May 2010

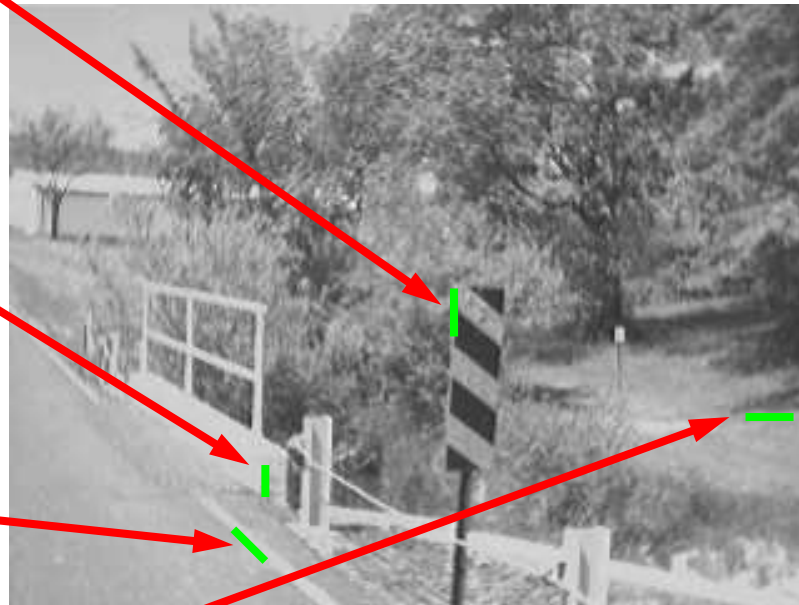


# Roadmap

- Robotic Vision & Challenges
- Camera Sensors
- Stereo Vision
- Primitive Features
- Segmentation
- Complex Features
- Shape Based Algorithms
- OpenCV Briefing
- Applications
- Unsolved Problems
- Homework

# What's an Edge?

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



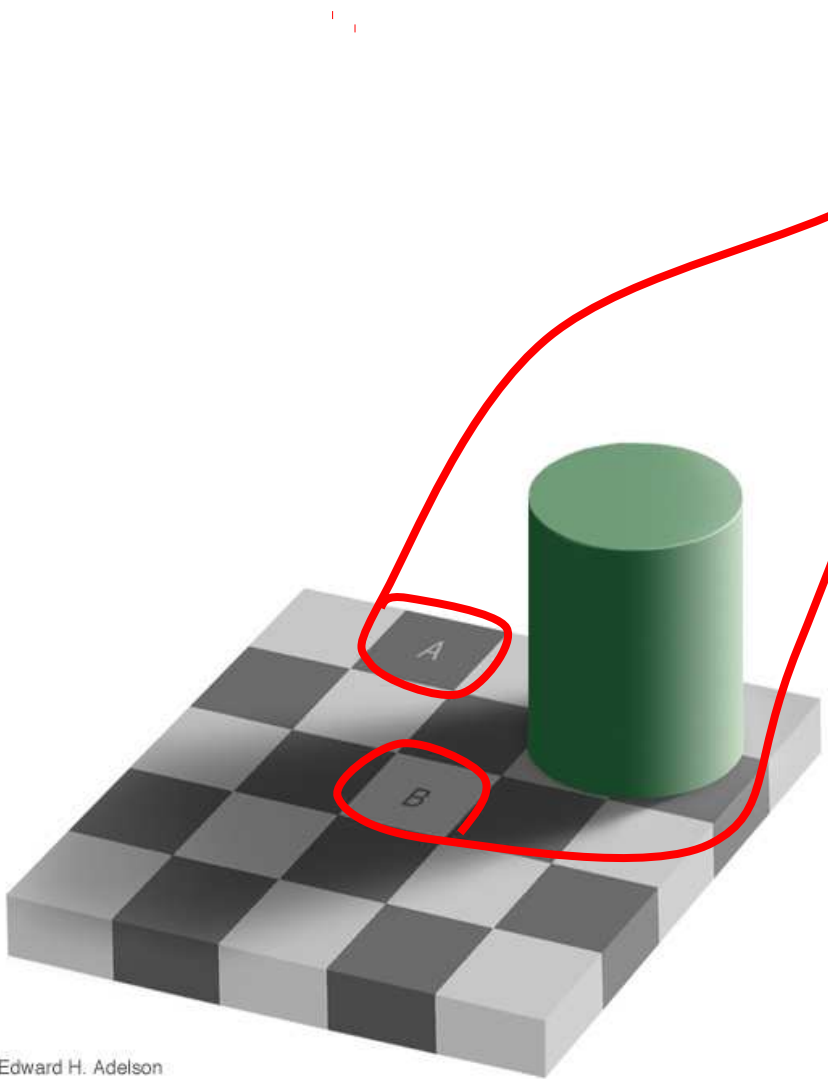
Slide credit: Christopher Rasmussen

# Invisible Surfaces

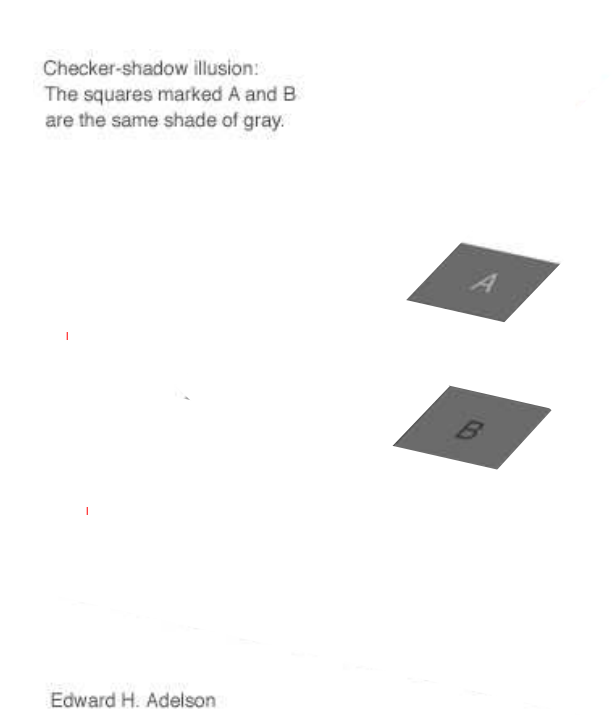


Bradski

# Lighting in 3D



Checker-shadow illusion:  
The squares marked A and B  
are the same shade of gray.



# Lighting -- Contrast

Which one is male and which one is female?

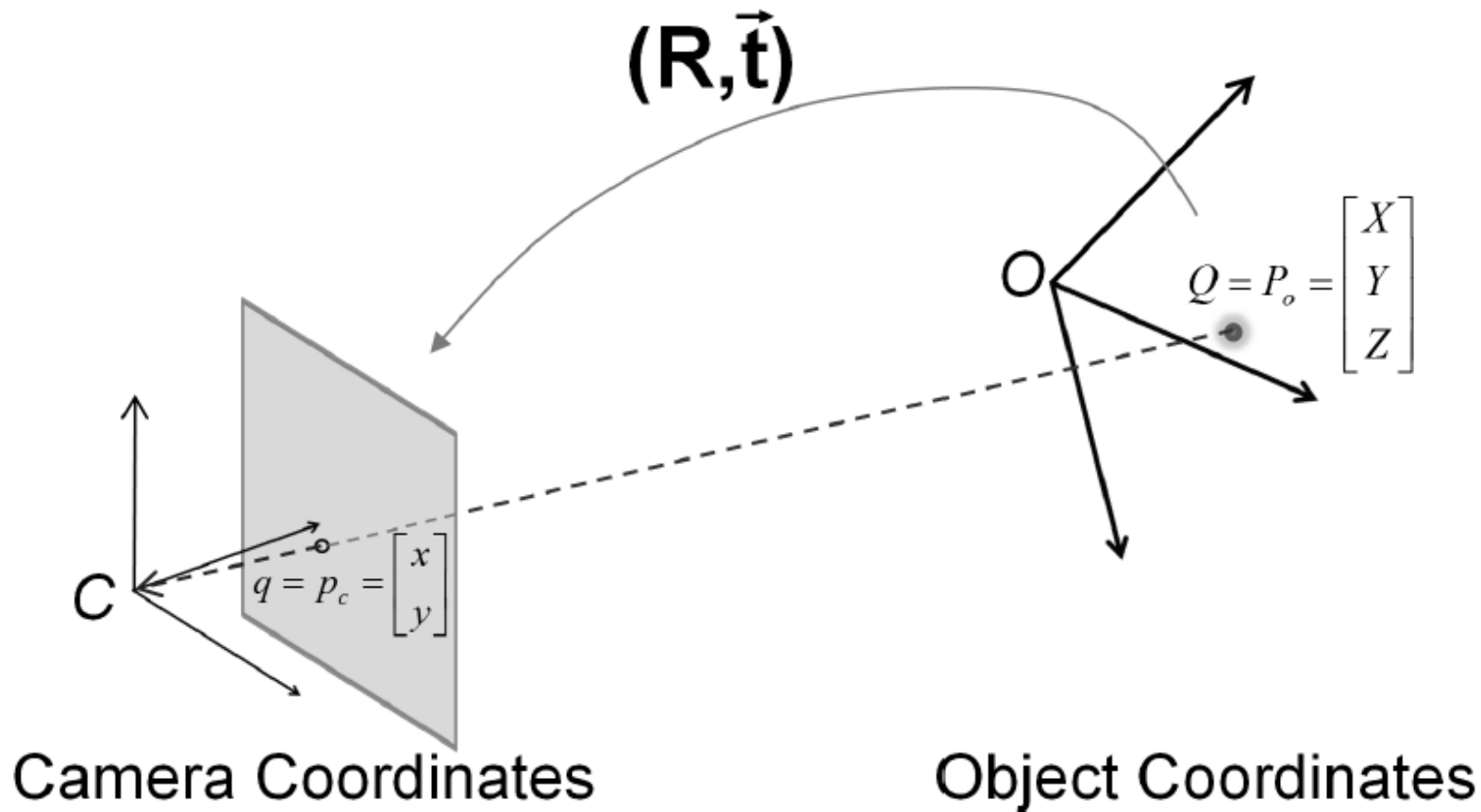


Illusion by: Richard Russell, Aarvard University

Gary Bradski, 2009

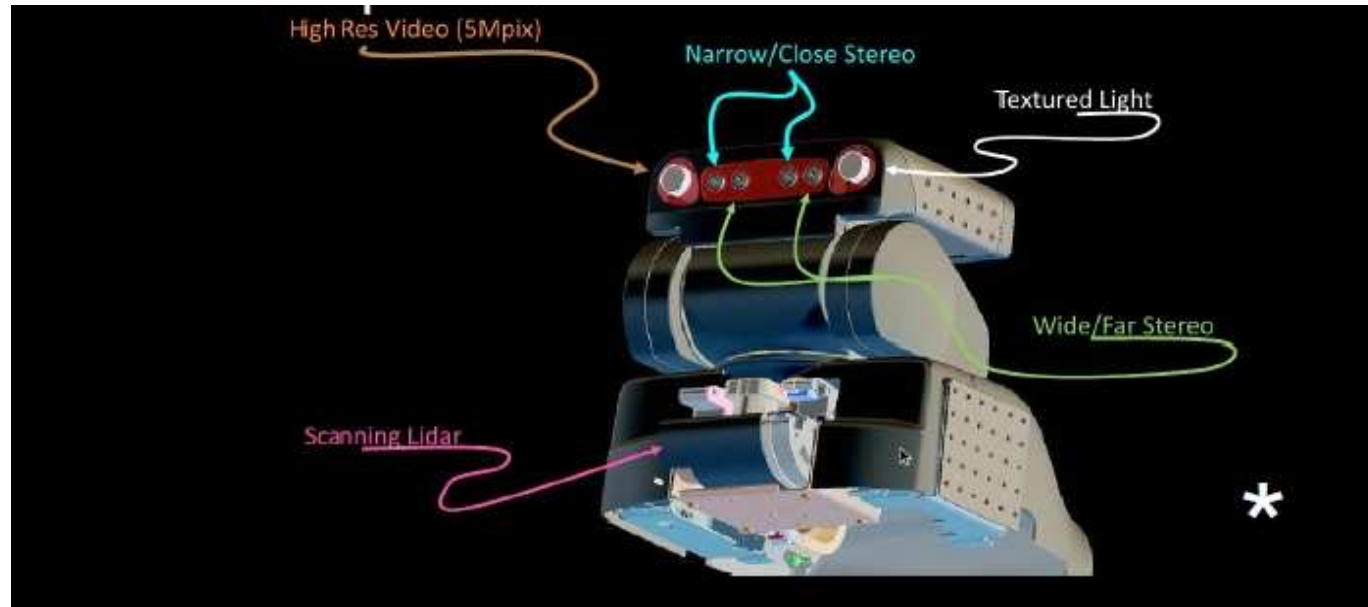
# Camera Sensors in Robotics

Pinhole camera model:



# Camera Sensors in Robotics

PR2 Head Setup:



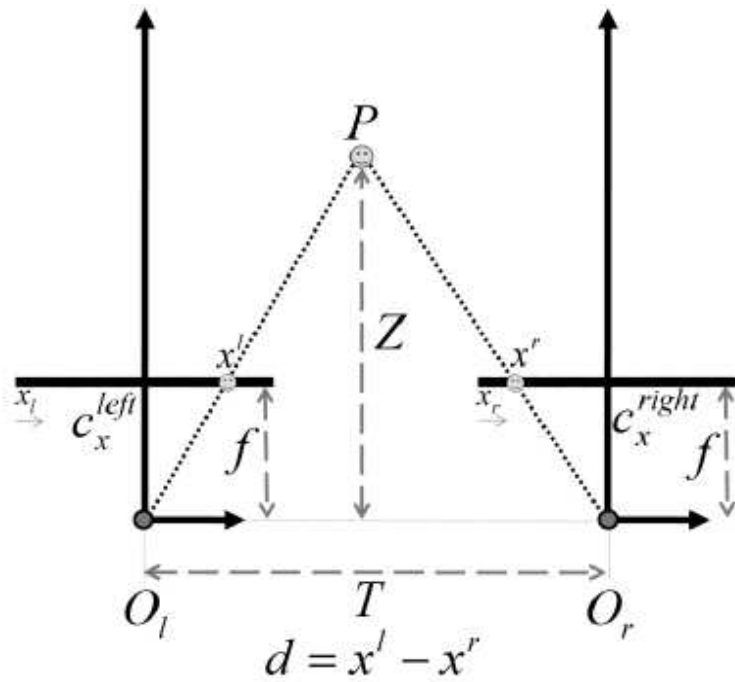
Willowgarage

Need to know about the camera sensor:

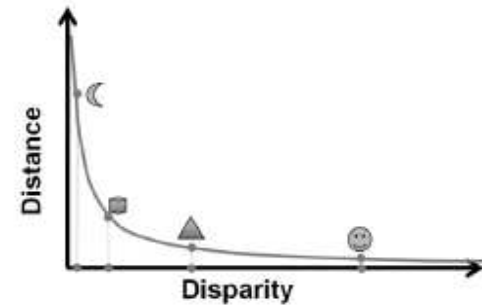
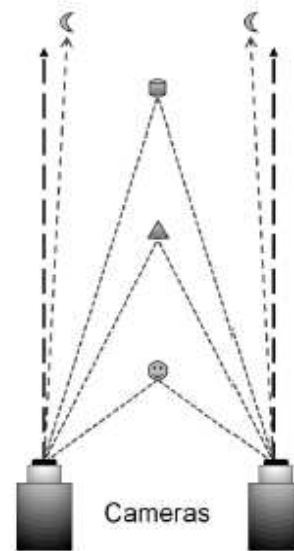
- ccd technology
- sensor size/resolution
- connection (usb, WiFi, FireWire, Ethernet)
- fps rate
- lenses (wide, narrow) and lens mount
- Mono, stereo, 3D

# Stereo

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



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



Bradski

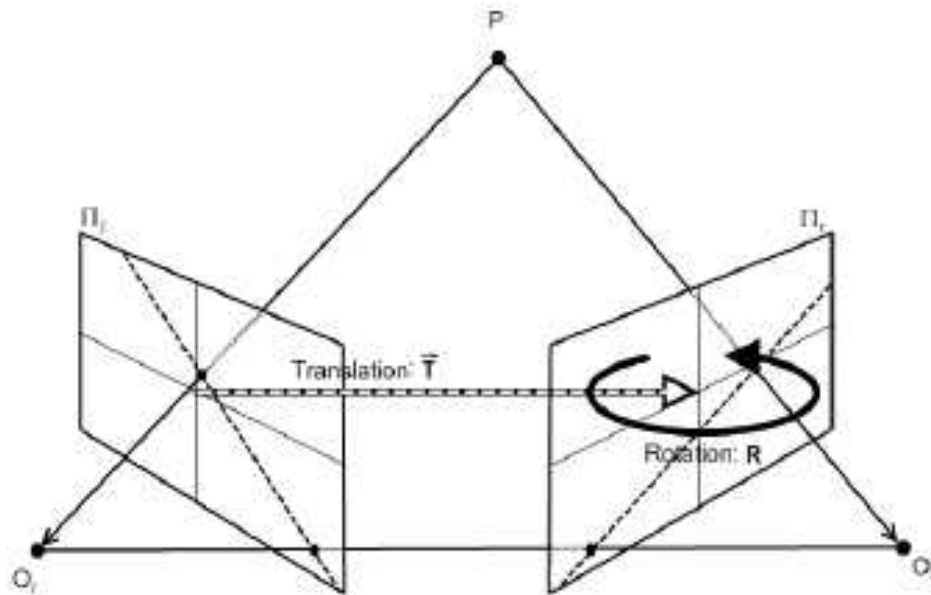
# Stereo

## Stereo Rectification

- We again collect many images of chessboards and use them to:
- Calibrate the left and right cameras
- Find the rotation and translation between them
- Mathematically align the cameras.

### Goal:

- Any given world point falls into the same row on the left and the right cameras.
- Called the Epipolar Constraint

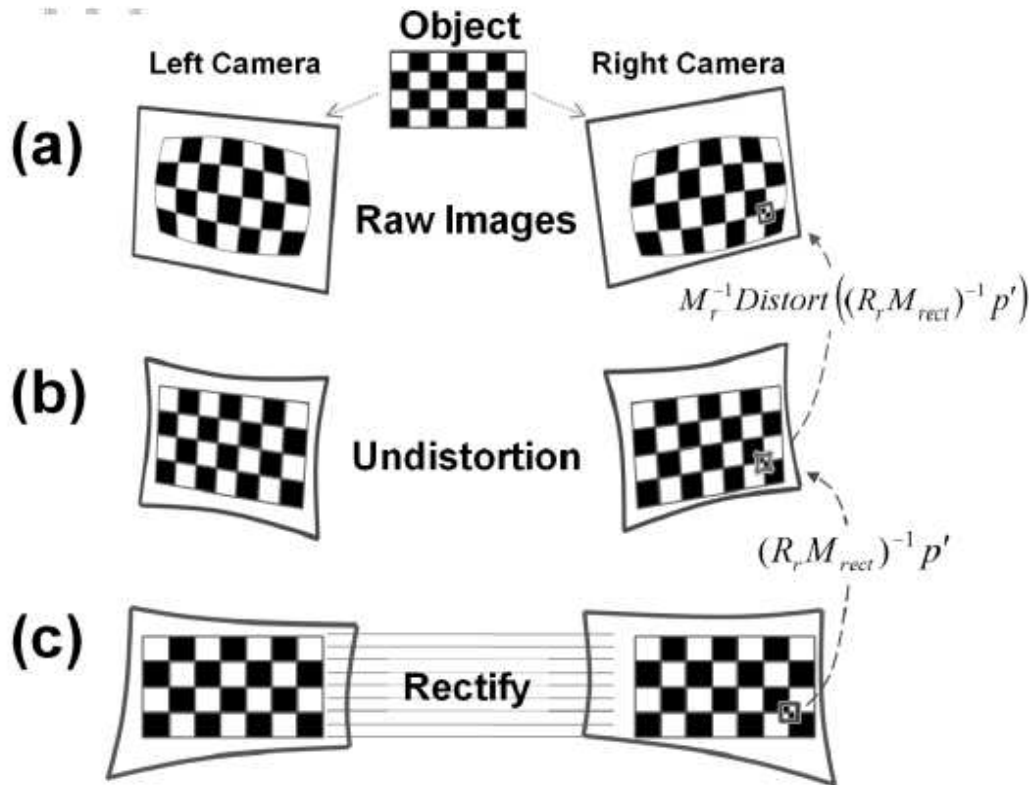


Bradski

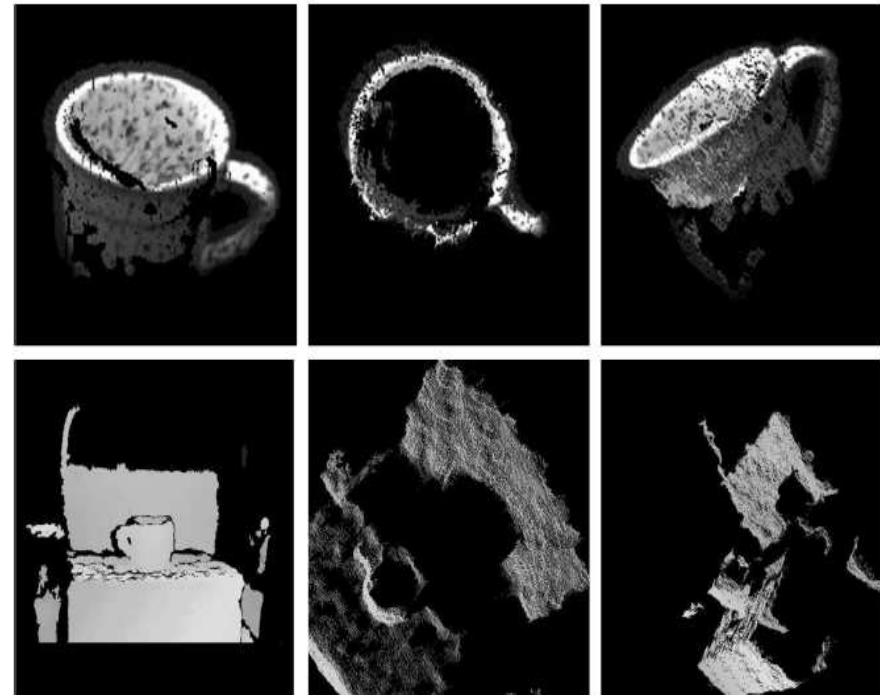
# Stereo

## Results:

### Calibration



### 3D Perception



Bradski

# Primitive Features: Edge Detector (Canny)

Kernels:

a)

|    |    |    |
|----|----|----|
| -1 | -1 | -1 |
| 2  | 2  | 2  |
| -1 | -1 | -1 |

b)

|    |   |    |
|----|---|----|
| -1 | 2 | -1 |
| -1 | 2 | -1 |
| -1 | 2 | -1 |

c)

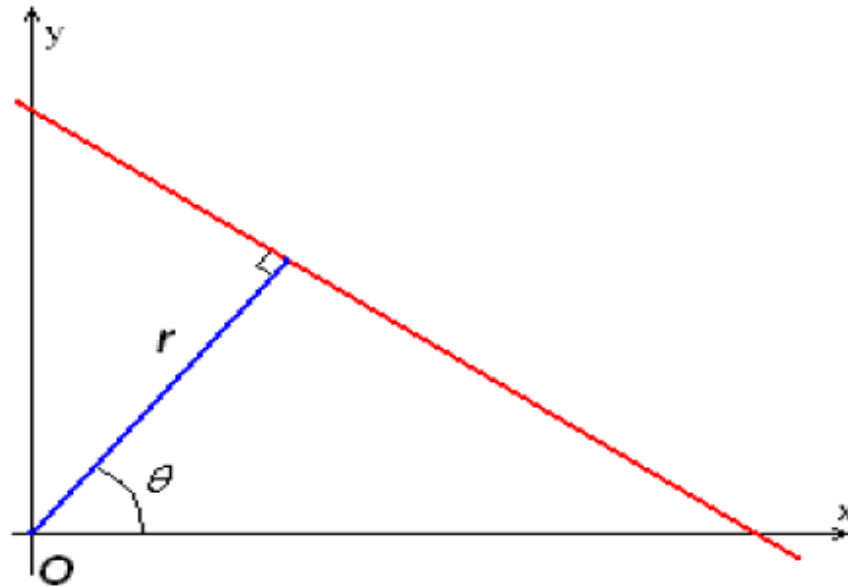
|    |    |    |
|----|----|----|
| -1 | -1 | 2  |
| -1 | 2  | -1 |
| 2  | -1 | -1 |

d)

|    |    |    |
|----|----|----|
| 2  | -1 | -1 |
| -1 | 2  | -1 |
| -1 | -1 | 2  |

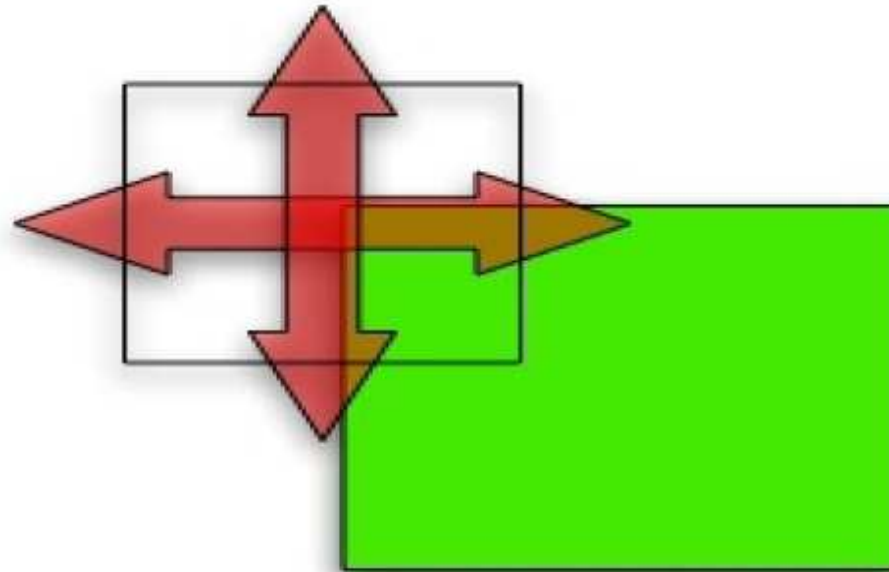


# Primitive Features: Line Detector (Hough Transform)



$$y = \left( -\frac{\cos \theta}{\sin \theta} \right) x + \left( \frac{r}{\sin \theta} \right)$$

# Primitive Features: Corner Detector (Harris)



$$A = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

# Segmentation

## Segmentation and Grouping

### What:

Segmentation breaks an image into groups over space and/or time

### Why:

- Motivation:
  - not for recognition
  - for compression
- Relationship of sequence/set of tokens
  - Always for a goal or application

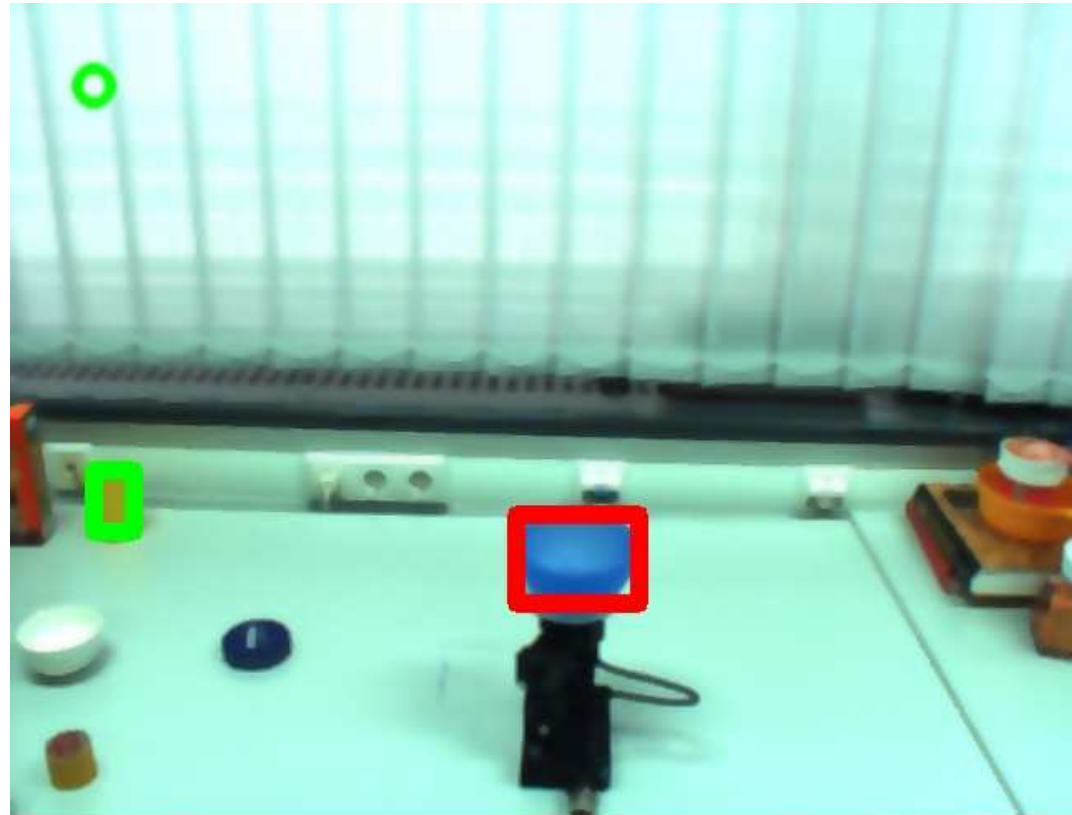
### Tokens are

- The things that are grouped (pixels, points, surface elements, etc., etc.)
- top down segmentation
  - tokens grouped because they lie on the same object
- bottom up segmentation
  - tokens belong together because of some local affinity measure
- Bottom up/Top Down need not be mutually exclusive

# Segmentation - Color

## Color Segmentation

- Optionally project the color space.
- Classify each pixel as one of up to 32 colors.
- Run length encode each scanline according to color.
- Group runs of the same color into regions.
- Pass over the structure gathering region statistics.
- Sort regions by color and size.



# Segmentation - Background

## Background Segmentation

1. Learn model of the background
  - By statistics (m,s); mixture of Gaussians; Adaptive filter, etc
2. Take absolute difference with current frame
  - Pixels greater than a threshold are candidate foreground
3. Use morphological open operation to clean up point noise.
4. Traverse the image and use flood fill to measure size of candidate regions.
  - Assign as foreground those regions bigger than a set value.
  - Zero out regions that are too small.
5. Track 3 temporal modes:
  - (1) Quick regional changes are foreground (people, moving cars);
  - (2) Changes that stopped a medium time ago are candidate background (chairs that got moved etc);
  - (3) Long term statistically stable regions are background.

# Segmentation - Background

## Background Techniques Compared

|                                | Moved Object | Time of Day                | Light Switch           | Waving Trees | Camouflage                        | Bootstrapping                | Foreground Aperture          |
|--------------------------------|--------------|----------------------------|------------------------|--------------|-----------------------------------|------------------------------|------------------------------|
| Test Image                     |              |                            |                        |              |                                   |                              |                              |
|                                | Chair moved  | Light gradually brightened | Light just switched on | Tree Waving  | Foreground covers monitor pattern | No clean background training | Interior motion undetectable |
| Ideal Foreground               |              |                            |                        |              |                                   |                              |                              |
| Adjacent Frame Difference      |              |                            |                        |              |                                   |                              |                              |
| Mean & Covariance [10]         |              |                            |                        |              |                                   |                              |                              |
| Mixture of Gaussians [3]       |              |                            |                        |              |                                   |                              |                              |
| Eigen-background [9]           |              |                            |                        |              |                                   |                              |                              |
| Linear Prediction [this paper] |              |                            |                        |              |                                   |                              |                              |
| Wallflower [this paper]        |              |                            |                        |              |                                   |                              |                              |

Bradski

# Segmentation – Graph Cuts

## Segmentation by Energy Minimization

$G = \{V, E\}$

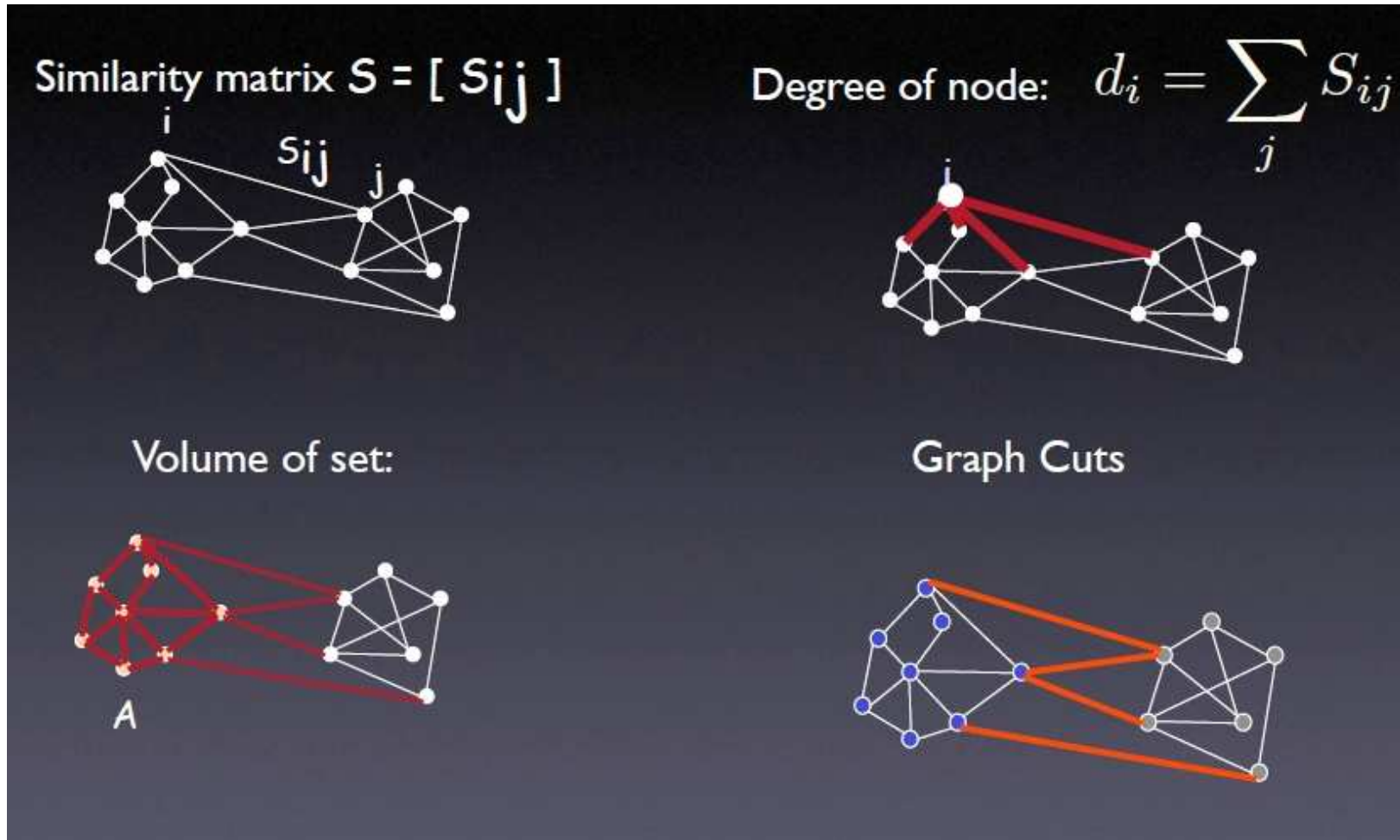
$V$ : graph nodes  
 $E$ : edges connection nodes

Image = { pixels }  
Pixel similarity

Shi

# Segmentation – Graph Cuts

Graph Terminology:



# Segmentation – Graph Cuts

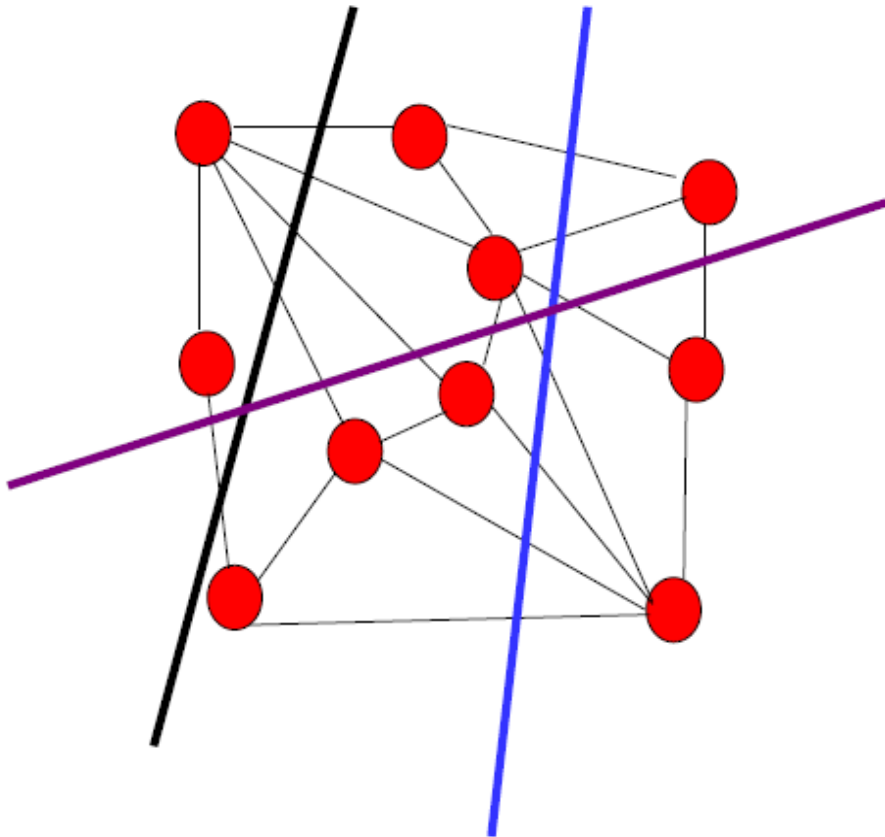
Useful Graph Algorithms:

- Minimal Spanning Tree
- Shortest path
- s-t Max. graph flow, Min. cut

# Segmentation – Graph Cuts

Minimum Cut:

A cut is minimum if the size of the cut is not larger than the size of any other cut.



A cut of a graph  $G$  is the set of edges  $S$  such that removal of  $S$  from  $G$  disconnects  $G$ .

Minimum cut is the cut of minimum weight, where weight of cut  $\langle A, B \rangle$  is given as

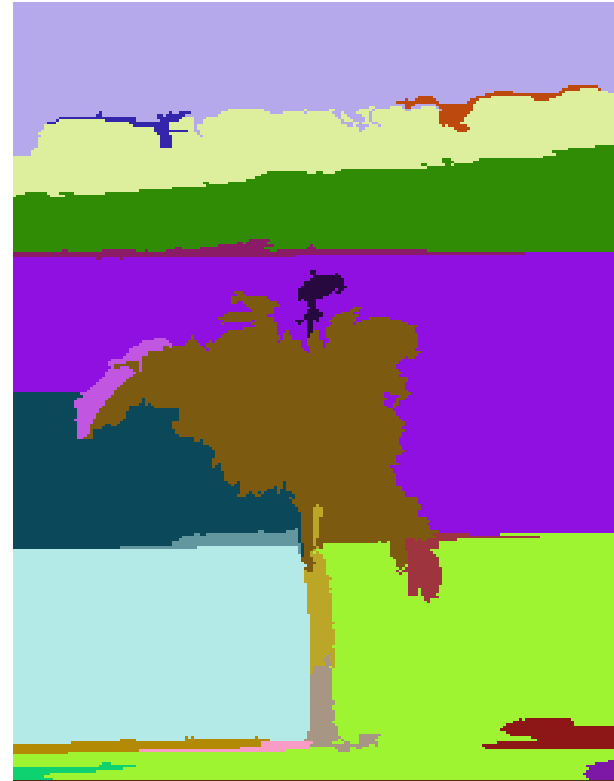
$$w(\langle A, B \rangle) = \sum_{x \in A, y \in B} w(x, y)$$

Bradski

H. Nagamochi, K. Nishimura and T. Ibaraki, “Computing all small cuts in an undirected network. SIAM J. Discrete Math. 10 (1997) 469-481.

# Segmentation – Graph Cuts

Result:



Felzenszwalb

Efficient Graph-Based Image Segmentation Pedro F.  
Felzenszwalb and Daniel P. Huttenlocher

# Complex Features – SIFT

What is it:

“For any object in an image, interesting points on the object can be extracted to provide a "feature description" of the object. This description, extracted from a training image, can then be used to identify the object when attempting to locate the object in a test image containing many other objects. It is important that the set of features extracted from the training image is robust to changes in image scale, noise, illumination, and local geometric distortion to perform reliable recognition”.

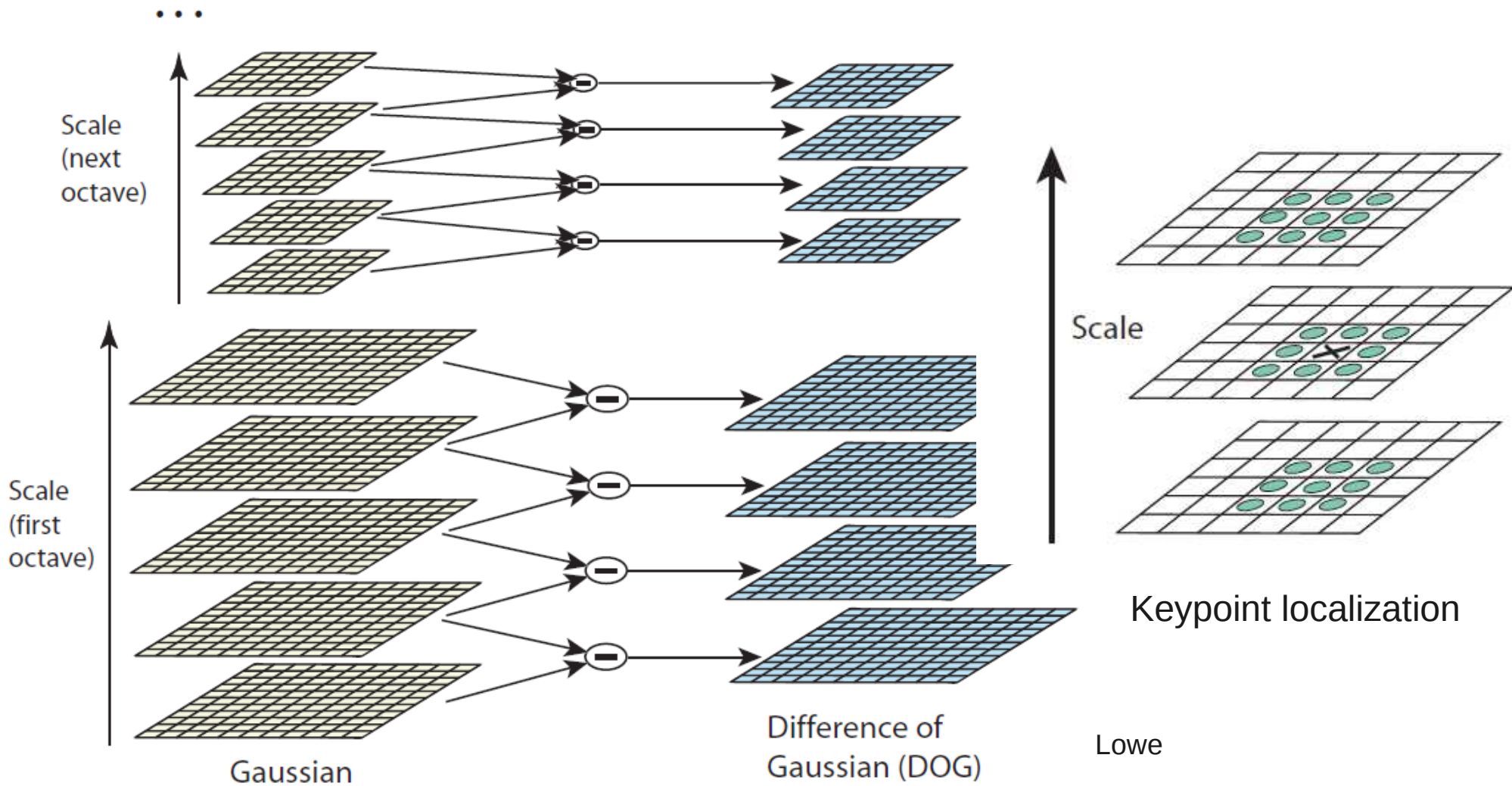
## **Key Features:**

- Scale-space extrema detection
- Keypoint localization
- Orientation assignment
- Keypoint descriptor

Distinctive image features from scale-invariant keypoints, Lowe, 2004

# Complex Features – SIFT

Scale-space extrema detection and Keypoint localization



# Complex Features – SIFT

Orientation assignment and Keypoint descriptor

For each candidate keypoint:

- Keypoints with low contrast are removed
- Responses along edges are eliminated
- The keypoint is assigned an orientation

**SIFT feature vector with  $4 \times 4 \times 8 = 128$  elements.**

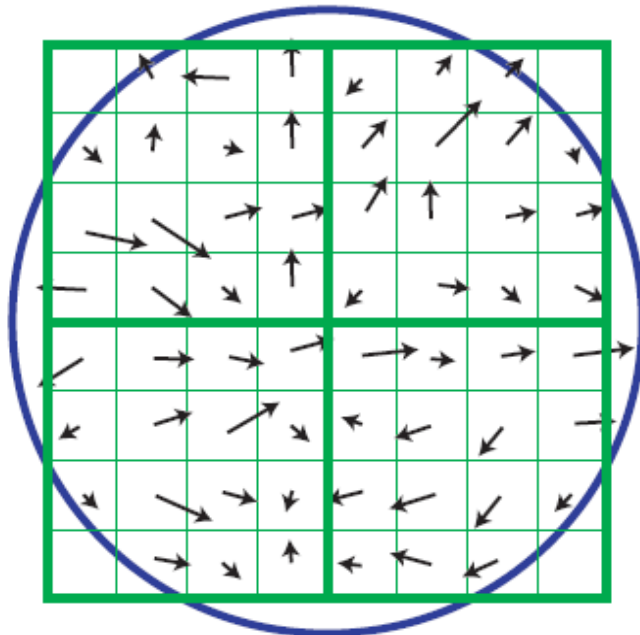
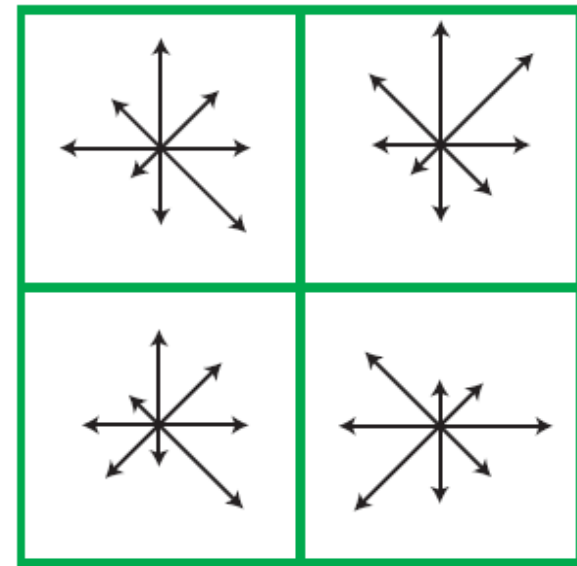


Image gradients



Keypoint descriptor

Lowe

# Complex Features – SIFT

Result:



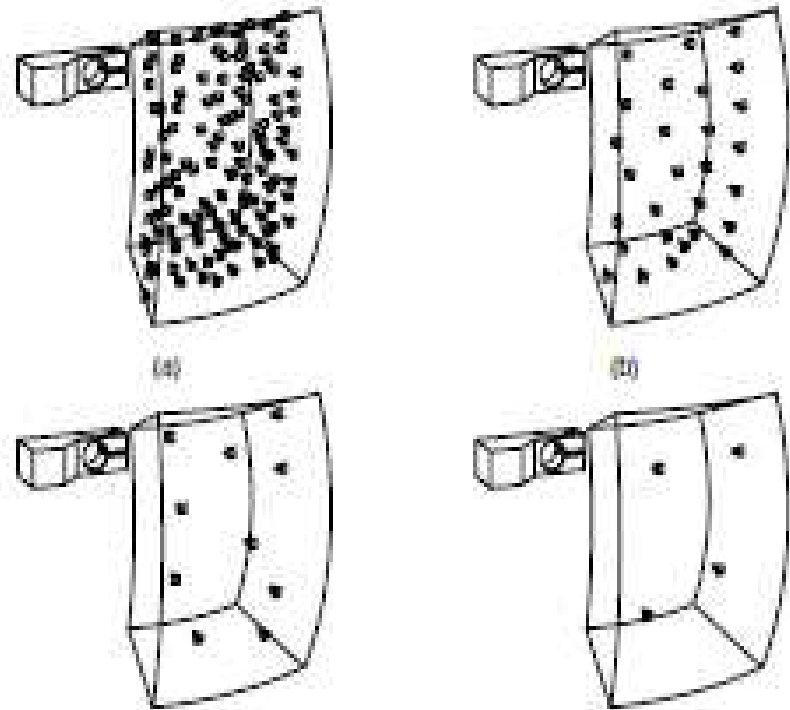
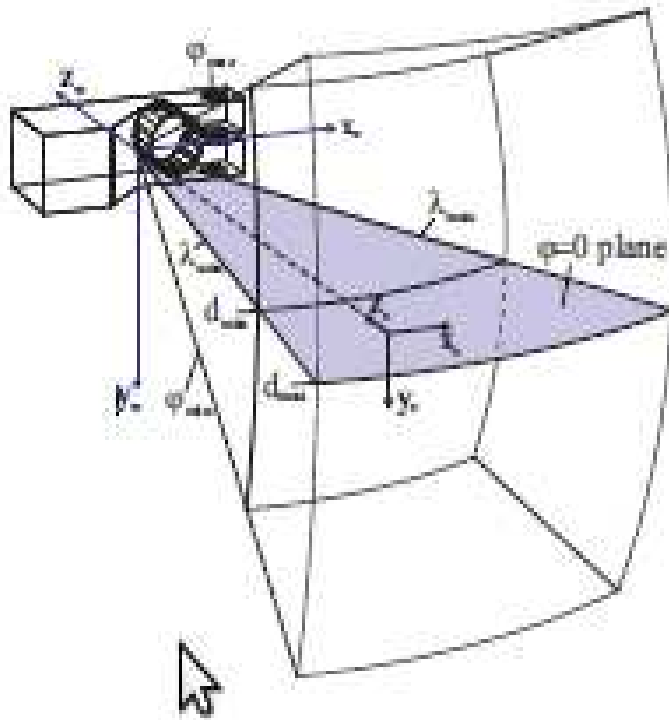
Low

## Complex Features – SIFT competing methods

- RIFT
- G-RIF
- SURF
- PCA-SIFT and GLOH
- Calonder descriptor
- Daisy (for dense stereo matching)
- KeyStar Detector

# Shape-based Algorithms – CAD Based Recognition of 3D Objects

- Hierarchical view-based approach that combines a pyramid search with a hierarchy of object views
- Image generation model that projects the geometry of a 3D model onto a color image from which a 2D model can be derived and matched against



# Shape-based Algorithms – CAD Based Recognition of 3D Objects

2D model similarity measure and results:

$$c = \frac{1}{n} \sum_{i=1}^n \frac{|\langle m_i, s_i \rangle|}{\|m_i\| \cdot \|s_i\|}$$



# Shape-based Algorithms – Chamfer Matching

Algorithm:

- Find strong edges in the image
- The distance transform of edges in an image is determined. The pixel value in a distance transform image is proportional to the distance of that pixel to the edge pixel closest to it.
- The model is then shifted over the distance transform image and at each shift position, the sum of distances at the model pixels is determined, and the shift position producing the smallest sum is chosen as the best-match position of the model in the image.

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

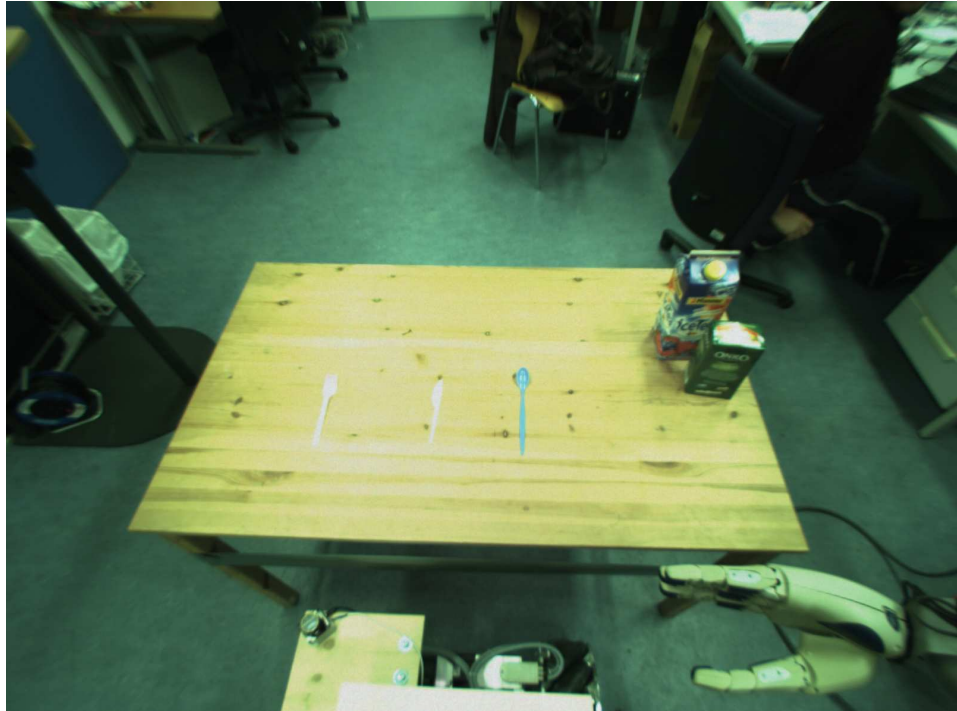
⇒

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 2 | 2 | 2 | 2 | 1 | 0 |
| 0 | 1 | 2 | 3 | 3 | 2 | 1 | 0 |
| 0 | 1 | 2 | 2 | 2 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

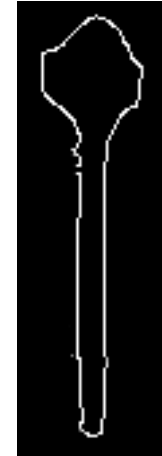
$$D_{Chess} = \max(|x_2 - x_1|, |y_2 - y_1|)$$

# Shape-based Algorithms – Chamfer Matching

Results:



+

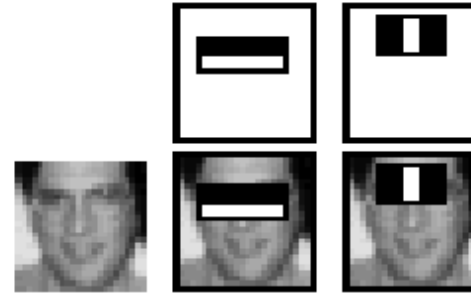
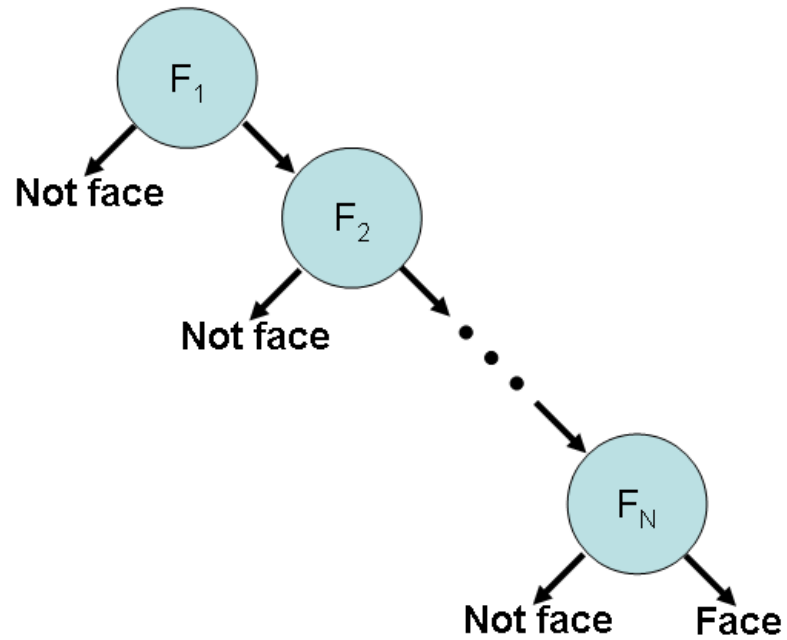


=



- The Contracting Curve Density Algorithm: Fitting Parametric Curve Models to Images Using Local Self-Adapting Separation Criteria, Hanek, 2004
- Dominant Orientation Templates, Hinterstoisser, 2010

# Applications – Face Detection



\*

Face Detection and Following:

**VIDEO:**  
`icub_following_face.mpeg`

# Applications in Robotics

Objects Detection:

VIDEO: [objectRecognition-22-04-08.avi](#)

# Applications in Robotics

SLAM (Simultaneous Localization and Mapping):

**VIDEO:**

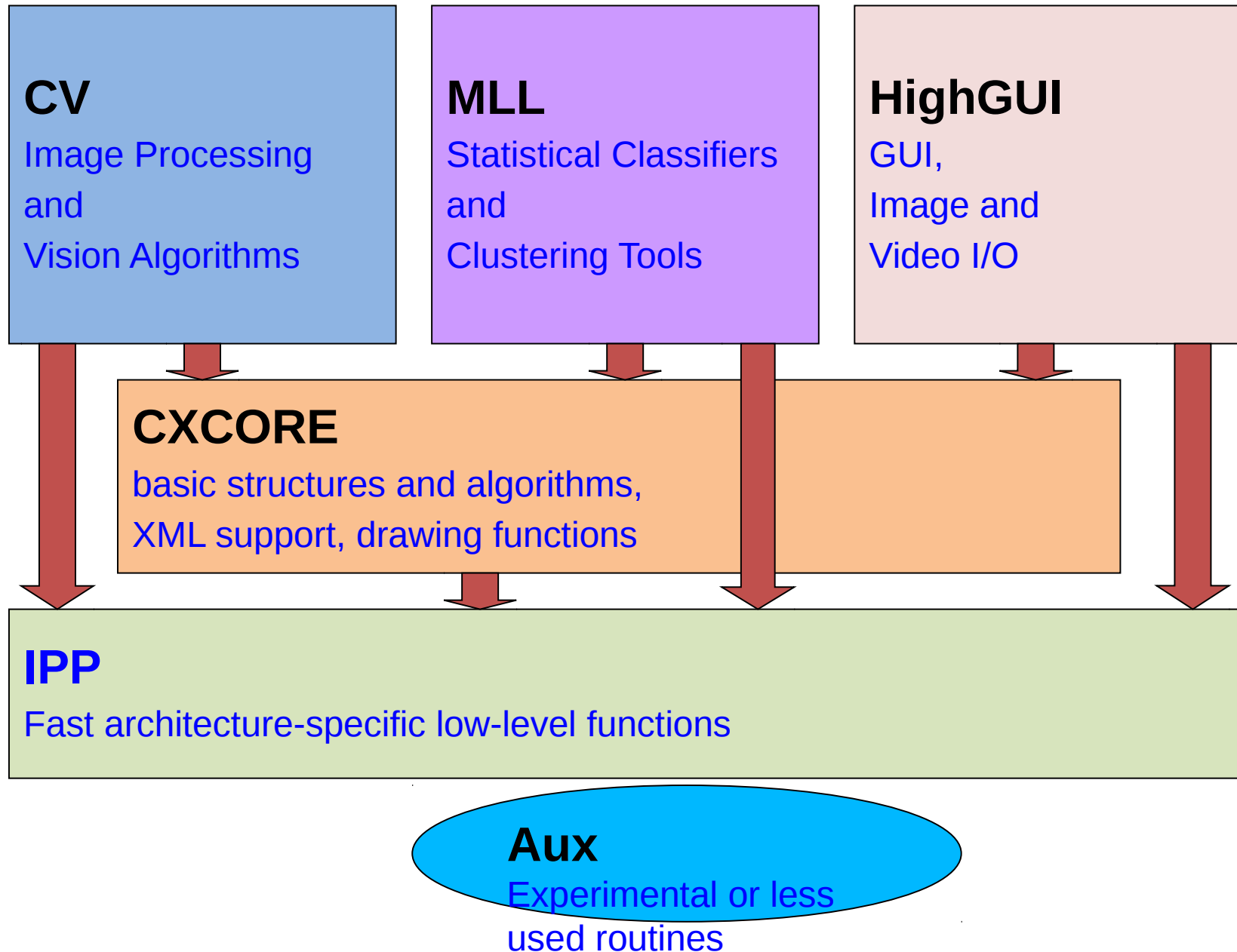
CircleHRP2.mpg

hrploopclose.mpg



# OpenCV

Intro (<http://opencv.willowgarage.com/wiki/>):



# OpenCV - IplImage

Basic image structure inherited from old IPL library

## Useful functions:

```
IplImage* cvCreateImage( vSize size, int depth,  
                        int channels );
```

```
void cvReleaseImage( IplImage** image );
```

```
IplImage* cvCloneImage( const IplImage* image );
```

```
void cvCopy( const CvArr* src, CvArr* dst,  
            const CvArr* mask=NULL );
```

```
void cvSetImageROI( IplImage* image, CvRect rect );
```

```
void cvResetImageROI( IplImage* image );
```

```
IplROI cvRectToROI( CvRect rect, int coi );
```

## IPL image header. Only important fields are shown in blue

```
typedef struct _IplImage  
{  
    int nSize;      /* sizeof(IplImage) */  
    int ID;        /* version (=0)*/  
    int nChannels;  /* Most of OpenCV functions support 1,2,3 or 4 channels */  
    int alphaChannel; /* ignored by OpenCV */  
    int depth;     /* pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S,  
IPL_DEPTH_16U, IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F and  
IPL_DEPTH_64F */  
    char colorModel[4]; /* ignored by OpenCV */  
    char channelSeq[4]; /* ditto */  
    int dataOrder; /* 0 - interleaved color channels, 1 - separate color channels.    int origin; /* 0 - top-left origin,    int align; /* Alignment of image rows (4 or 8).    int width; /* image width in pixels */  
    int height; /* image height in pixels */  
    struct _IplROI *roi; /* image ROI. when it is not NULL, this specifies image region to    struct _IplImage *maskROI; /* must be NULL in OpenCV */  
    void *imgeld; /* ditto */  
    struct _IplTileInfo *tileInfo; /* ditto */  
    int imageSize; /* image data size in bytes */  
    char *imageData; /* pointer to aligned image data */  
    int widthStep; /* size of aligned image row in bytes */  
    int BorderMode[4]; /* border completion mode, ignored by OpenCV */  
    int BorderConst[4]; /* ditto */  
    char *imageDataOrigin; /* pointer to a very origin of image data} IplImage;
```

# OpenCV - IplImage

## # Create and position a window:

```
cvNamedWindow("win1", CV_WINDOW_AUTOSIZE);  
cvMoveWindow("win1", 100, 100); // offset from the UL corner of the screen
```

## # Load an image:

```
IplImage* img=0;  
img=cvLoadImage(fileName);  
if(!img) printf("Could not load image file: %s\n",fileName);
```

## # Display an image:

```
cvShowImage("win1",img);
```

Can display a color or grayscale byte/float-image. A byte image is assumed to have values in the range  $[0..255]$ . A float image is assumed to have values in the range  $[0..1]$ . A color image is assumed to have data in BGR order.

## # Close a window:

```
cvDestroyWindow("win1");
```

## # Resize a window:

```
cvResizeWindow("win1",100,100); // new width/height in pixels
```

# OpenCV - I/O Images Allocating and Release

## # Allocate an image:

```
IpImage* cvCreateImage(CvSize size, int depth, int
channels);
  size:
cvSize(width,height);
  depth:
pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S,
IPL_DEPTH_16U, IPL_DEPTH_16S,
IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F
  channels:
Number of channels per pixel. Can be 1, 2, 3 or 4. The
channels are interleaved. The usual data layout of a
color image is
  b0 g0 r0 b1 g1 r1 ...
```

## Examples:

```
// Allocate a 1-channel byte image
IpImage*
img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8
U,1);
// Allocate a 3-channel float image
IpImage*
img2=cvCreateImage(cvSize(640,480),IPL_DEPTH_32
F,3);
# Release an image:
IpImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,
1);
cvReleaseImage(&img);
```

## # Clone an image:

```
IpImage*
img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8
U,1);
IpImage* img2;
img2=cvCloneImage(img1);
```

## # Set/get the region of interest:

```
void cvSetImageROI(IpImage* image, CvRect rect);
void cvResetImageROI(IpImage* image);
vRect cvGetImageROI(const IpImage* image);
```

**All** OpenCV functions support ROI.

## # Set/get the channel of interest:

```
void cvSetImageCOI(IpImage* image, int coi); // 0=all
int cvGetImageCOI(const IpImage* image);
```

The majority of OpenCV functions do NOT support COI.

# OpenCV - I/O Reading and Writing Images

## # Reading an image from a file:

```
IplImage* img=0;  
img=cvLoadImage(fileName);  
if(!img) printf("Could not load image file: %s\n",fileName);
```

### Supported image formats:

BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF, TIF

**By default, the loaded image is forced to be a 3-channel color image. This default can be modified by using:**

```
img=cvLoadImage(fileName,flag);
```

flag: >0 the loaded image is forced to be a 3-channel color image  
=0 the loaded image is forced to be a 1 channel grayscale image  
<0 the loaded image is loaded as is (with number of channels in the file).

## # Writing an image to a file:

```
if(!cvSaveImage(outFileName,img)) printf("Could not save: %s\n",outFileName);
```

**The output file format is determined based on the file name extension.**

# OpenCV - Drawing into an Image

## # Draw a box:

```
// draw a box with red lines of width 1 between (100,100) and (200,200)
cvRectangle(img, cvPoint(100,100), cvPoint(200,200), cvScalar(255,0,0), 1);
```

## # Draw a circle:

```
// draw a circle at (100,100) with a radius of 20. Use green lines of width 1
cvCircle(img, cvPoint(100,100), 20, cvScalar(0,255,0), 1);
```

## # Draw a line segment:

```
// draw a green line of width 1 between (100,100) and (200,200)
cvLine(img, cvPoint(100,100), cvPoint(200,200), cvScalar(0,255,0), 1);
```

## # Draw a set of polylines:

```
CvPoint curve1[]={10,10, 10,100, 100,100, 100,10};
CvPoint curve2[]={30,30, 30,130, 130,130, 130,30, 150,10};
CvPoint* curveArr[2]={curve1, curve2};
int nCurvePts[2]={4,5};
int nCurves=2;
int isCurveClosed=1;
int lineWidth=1;

cvPolyLine(img,curveArr,nCurvePts,nCurves,isCurveClosed,cvScalar(0,255,255),lineWidth);
```

## # Draw a set of filled polygons:

```
cvFillPoly(img,curveArr,nCurvePts,nCurves,cvScalar(0,255,255));
```

# OpenCV - Working with Video

## Capturing a frame from a video sequence

\* OpenCV supports capturing images from a camera or a video file (AVI).

\* Initializing capture from a camera:

```
CvCapture* capture = cvCaptureFromCAM(0); // capture from video device #0
```

\* Initializing capture from a file:

```
CvCapture* capture = cvCaptureFromAVI("infile.avi");
```

\* Capturing a frame:

```
IplImage* img = 0;  
if(!cvGrabFrame(capture)){           // capture a frame  
    printf("Could not grab a frame\n\n");  
    exit(0);  
}  
img=cvRetrieveFrame(capture);         // retrieve the captured frame
```

To obtain images from several cameras simultaneously, first grab an image from each camera. Retrieve the captured images after the grabbing is complete.

\* Releasing the capture source:

```
cvReleaseCapture(&capture);
```

Note that the image captured by the device is allocated/released by the capture function. There is no need to release it explicitly.

# OpenCV examples

In samples\c

bgfg\_codebook.cpp - Use of a image value codebook for background detection for collectin objects  
bgfg\_segm.cpp - Use of a background learning engine  
blobtrack.cpp - Engine for blob tracking in images  
calibration.cpp - Camera Calibration  
camshiftdemo.c - Use of meanshift in simple color tracking  
contours.c - Demonstrates how to compute and use object contours  
convert\_cascade.c - Change the window size in a recognition cascade  
convexhull.c - Find the convex hull of an object  
delaunay.c - Triangulate a 2D point cloud  
demhist.c - Show how to use histograms for recognition  
dft.c - Discrete fourier transform  
distrans.c - distance map from edges in an image  
drawing.c - Various drawing functions  
edge.c - Edge detection  
facedetect.c - Face detection by classifier cascade  
ffilldemo.c - Flood filling demo  
find\_obj.cpp - Demo use of SURF features  
fitellipse.c - Robust elipse fitting  
houghlines.c - Line detection  
image.cpp - Shows use of new image class, CvImage();  
inpaint.cpp - Texture infill to repair imagery  
kalman.c - Kalman filter for trackign  
kmeans.c - K-Means  
laplace.c - Convolve image with laplacian.

letter\_recog.cpp - Example of using machine learning Boosting, Backpropagation (MLP) and Random forests  
lkdemo.c - Lukas-Canada optical flow  
minarea.c - For a cloud of points in 2D, find min bounding box and circle. Shows use of Cv\_SEQ  
morphology.c - Demonstrates Erode, Dilate, Open, Close  
motempl.c - Demonstrates motion templates (orthogonal optical flow given silhouettes)  
mushroom.cpp - Demonstrates use of decision trees (CART) for recognition  
pyramid\_segmentation.c - Color segmentation in pyramid  
squares.c - Uses contour processing to find squares in an image  
stereo\_calib.cpp - Stereo calibration, recognition and disparity map computation  
watershed.cpp - Watershed transform demo.

# OpenCV and ROS

## ROS sensor\_msgs/Image:

[sensor\_msgs/Image]:

Header header

uint32 seq

time stamp

string frame\_id

uint32 height

uint32 width

string encoding

uint8 is\_bigendian

uint32 step

uint8[] data

Go to code: [openCv\\_to\\_ros.cpp](#)

# Homework

- 1) Download package sie\_three: [http://ias.cs.tum.edu/~pangerci/sie\\_three.tar](http://ias.cs.tum.edu/~pangerci/sie_three.tar)
- 2) Write a node converter.cpp that loads the image scene.png from disk, converts it into the grayscale image, resizes it for 50%, draws one rectangle and one circle into it and saves it back to disk under the name scene\_grayscale.png.
- 3) Familiarize yourself with the code in the node src/openCV\_to\_ros.cpp and learn how to load the scene.png image and publish it on a topic.
- 4) Write a node find\_color.cpp that pulls the image (scene.png) from the topic openCV\_to\_ros.cpp publishes to, and finds the red object on the table. Draw a rectangle around the object in the original image and save the image under the name scene\_red\_object.png. Hint: see the example code in src/blob\_detection\_hsv.cpp or check out the cmvision library in ROS: <http://www.ros.org/doc/api/cmvision/html/>
- 5) Write a node find\_edges.cpp that pulls the image (scene.png) from the topic openCV\_to\_ros.cpp publishes to, and finds all the edges in it. Hint: see the code in vision\_opencv/opencv2/build/opencv-svn/samples/c/edge.c. Save the image with edges under the name scene\_edges.png
- 6) Bonus: SURF (alternative to SIFT) descriptor. Study the code in vision\_opencv/opencv2/build/opencv-svn/samples/c/find\_obj.cpp. Take 2 images of one object in your household from different viewpoints and carry out matching on them. Save results under the name my\_object\_surf.png.
- 7) Send all the code and images to [blodow@cs.tum.edu](mailto:blodow@cs.tum.edu) by 20.5.2010.

# Unsolved Issues

Transparent Objects and Objects with Reflective Surface:

