Computer Vision for AIBOs

V.V. Hafner, 05/2006

AIBO Tech Specs (inkl. Kamera)

Teaching a New Dog Old Tricks

Erik Andersen

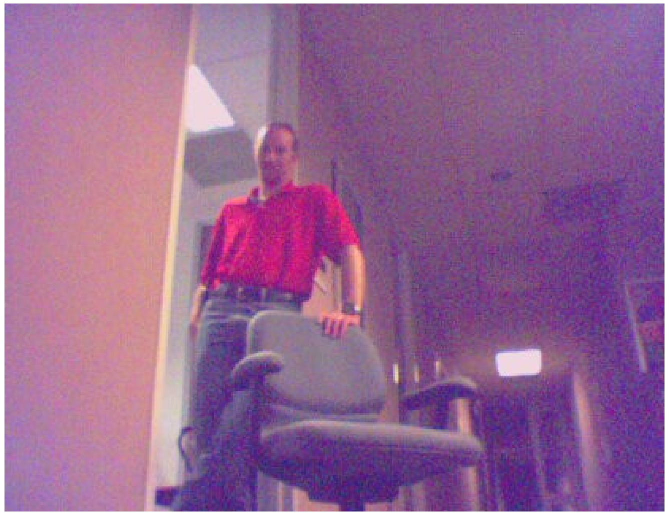
The AIBO

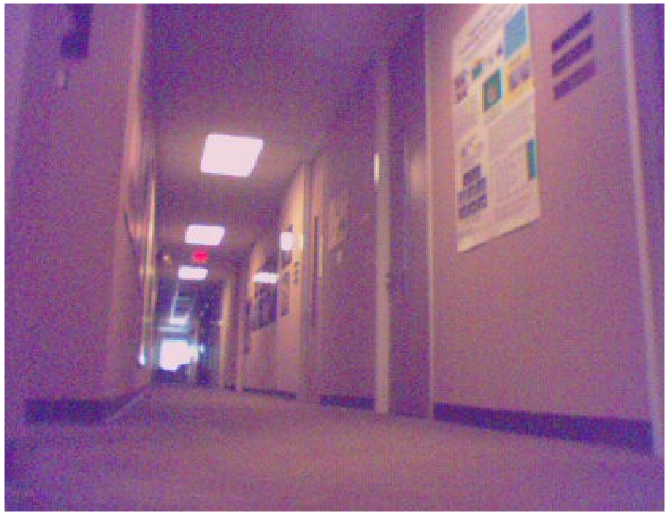
- Sony Entertainment Robot
- Released in 1999
- First production run: 3000 in Japan, 2000 in US; sold out in Japan in 20 minutes and in US in four days
- $2000 – but good specs
- Connects through wireless networks
- Acts like a dog: chases its ball, finds its bone, etc.

AIBO’s Nuts and Bolts

- 576 MHz MIPS Processor
- 64 MB RAM
- 3 Infrared Sensors:
  - Short-range head sensor (Range: 50-500 mm)
  - Long-range head sensor (200-1500 mm)
  - “Emergency” chest sensor (100-900 mm)
- 416x320 pixel nose-mounted camera
- Several joints and degrees of freedom

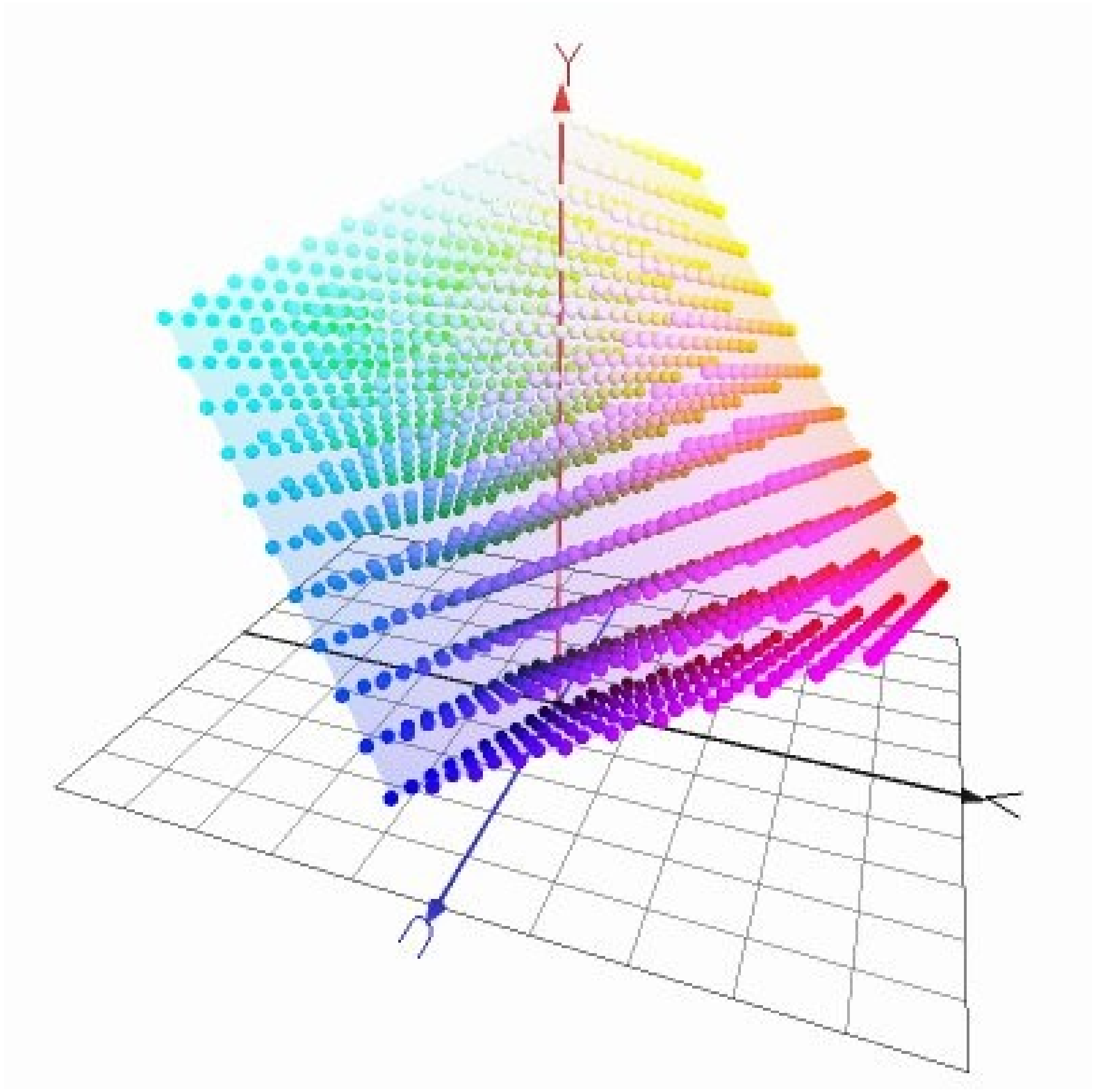
From AIBO’s Point of View



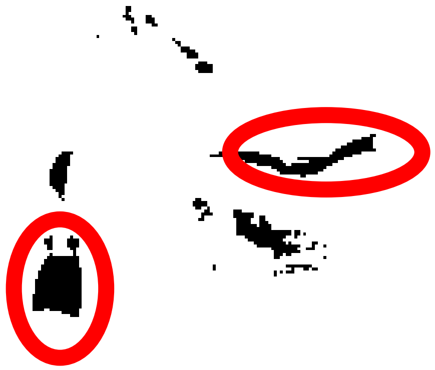


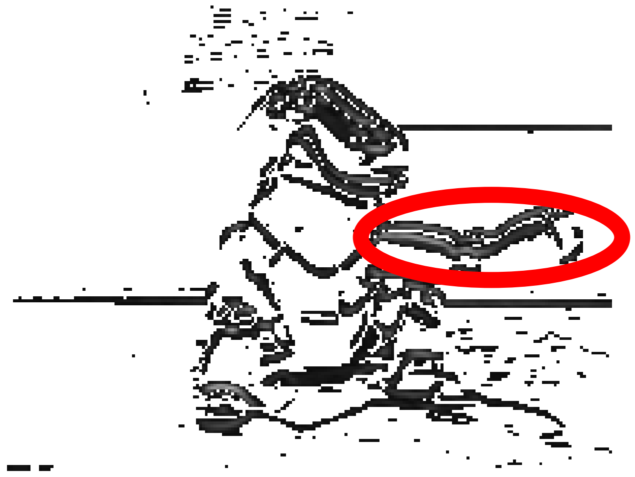
YUV (Luminance, Chrominance)

- Similar to YIQ and YCbCr
- Used for the PAL and SECAM broadcast television system
- Amount of information needed to define a color is greatly reduced

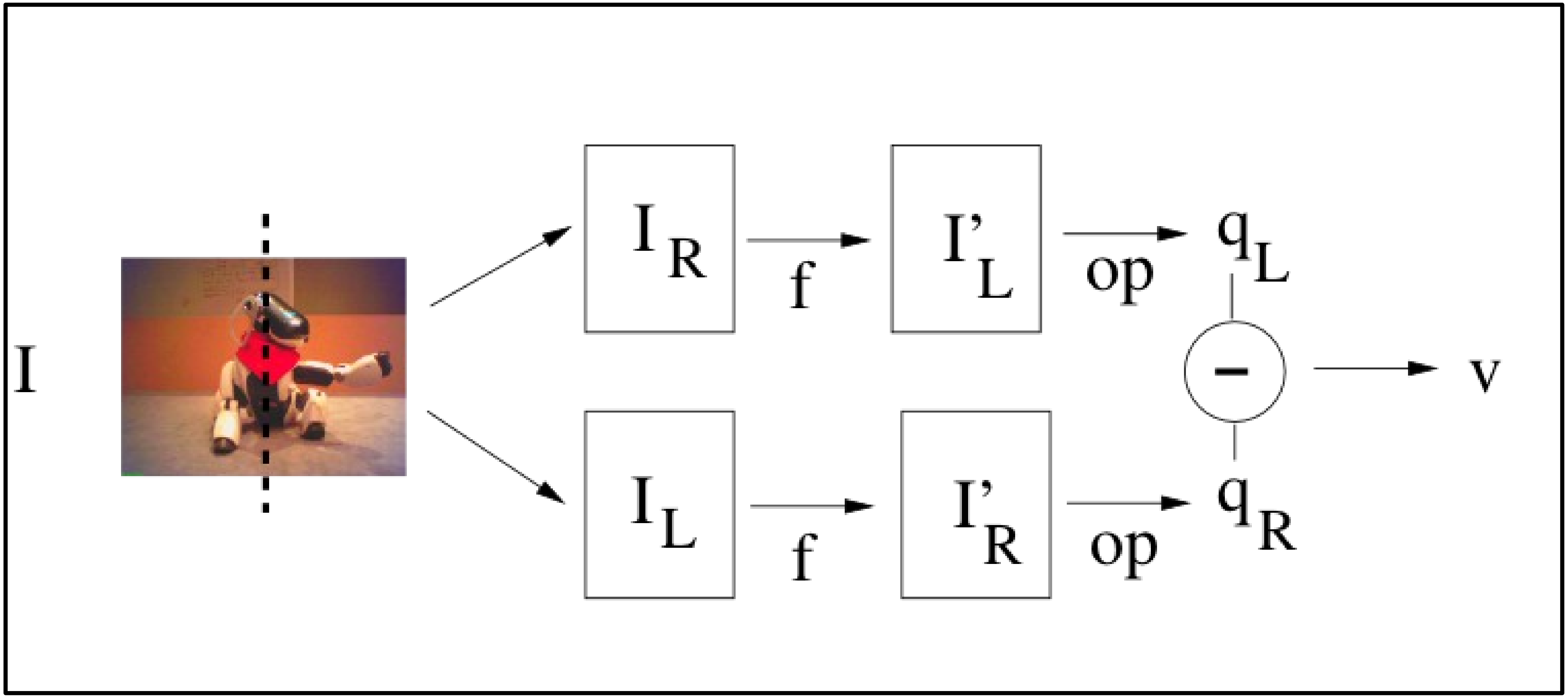


Helligkeits- und Kantendetektor





Beispiel für Pointing Detektion



URBI Camera Device

URBI camera device

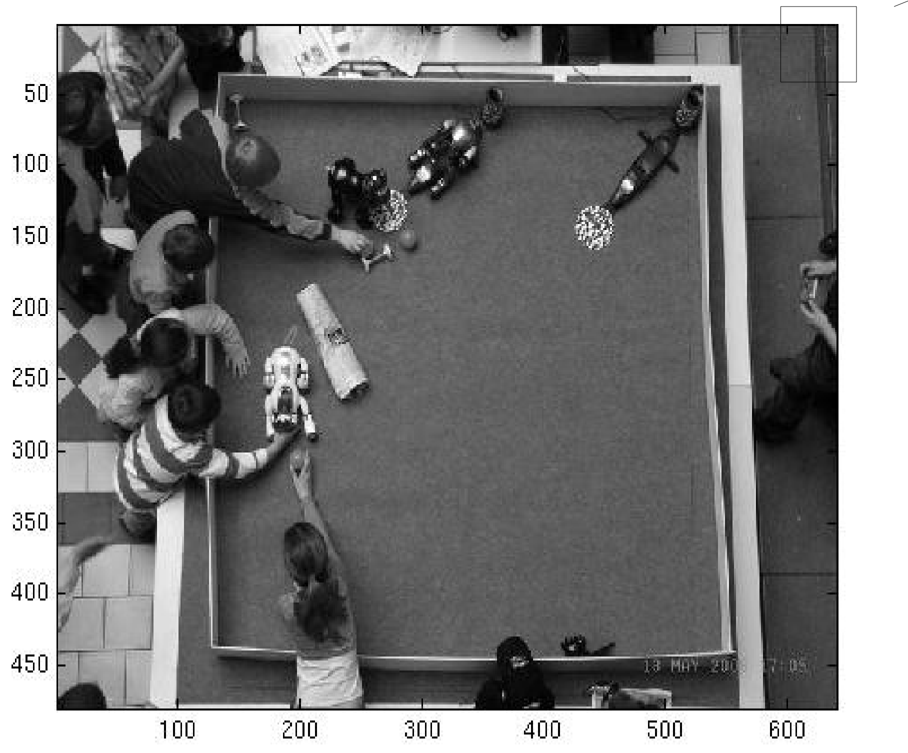
• camera.shutter :	the camera shutter speed: 1=SLOW (default), 2=MID, 3=FAST
• camera.gain :	the camera gain: 1=LOW, 2=MID, 3=HIGH (default)
• camera.wb :	the camera white balance: 1=INDOOR (default), 2=OUTDOOR,3=FLUO
• camera.format :	the camera image format: 0=YCbCr 1=jpeg (default)
• camera.jpegfactor :	the jpeg compression factor (0 to 100). Default=80
• camera.resolution :	the image resolution: 0:208x160 (default) 1:104x80 2:52x40
• camera.reconstruct :	reconstruction of the high resolution image(slow): 0:no (default) 1:yes
• camera.width :	image width
• camera.height :	image height
• camera.xfov :	camera x Field Of View (degrees)
• camera.yfov :	camera y Field Of View (degrees)

The Ball Soft Device

Detecting a ball involves image processing and cannot be written directly in URBI for obvious efficiency reasons. The best way to provide such components (like visual processing or sound processing) is to write a soft device in C++, Java or Matlab, and to interface it in URBI. We will not describe at this stage how to write such a "soft" device, but instead we will already use one: the ball soft device.

The ball soft device is directly integrated in the Aibo URBI server and you can use it directly, just like any physical device. It has no ball.val variable but it has a **ball.x** and **ball.y** variable which are equal to the coordinate of the ball in the image between -1/2 and 1/2. When there is a ball visible, **ball.visible** is equal to 1, zero otherwise. It also have a **ball.size** variable which give the size of the ball in the image, expressed as a number of pixels. These simple soft device variables are already enough to do many interesting applications, as we will see below.

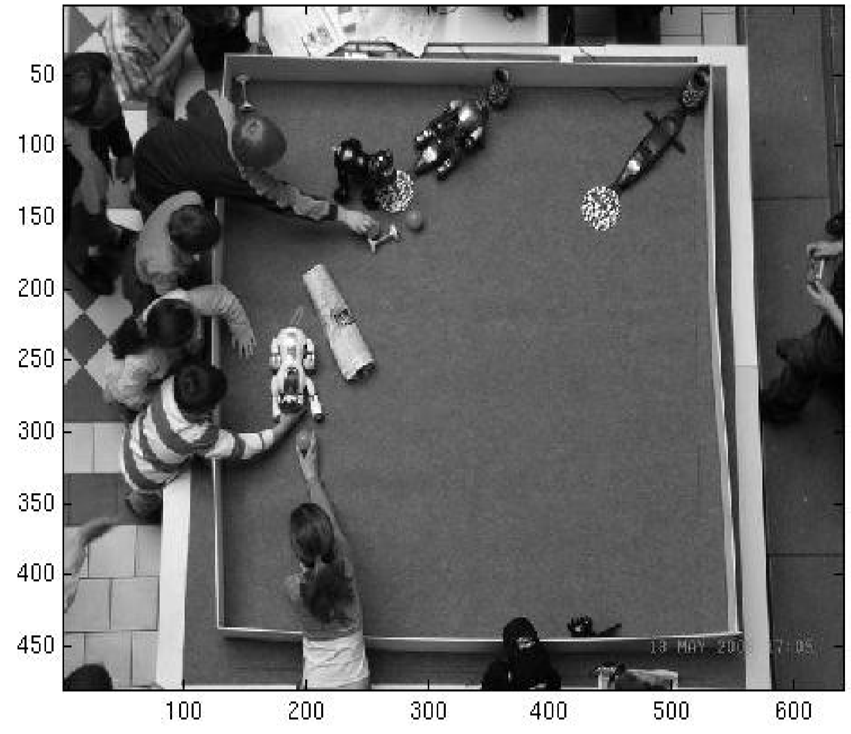
Kantenerkennung – Methoden und Beispiele

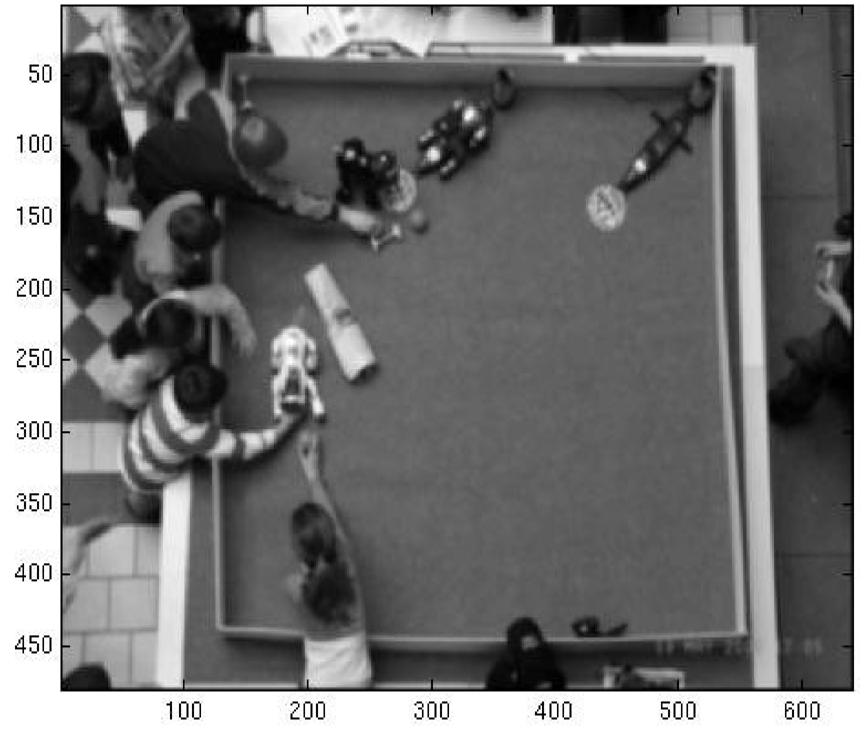


1	2	1
2	4	2
1	2	1

Convolution Filter

Smoothing



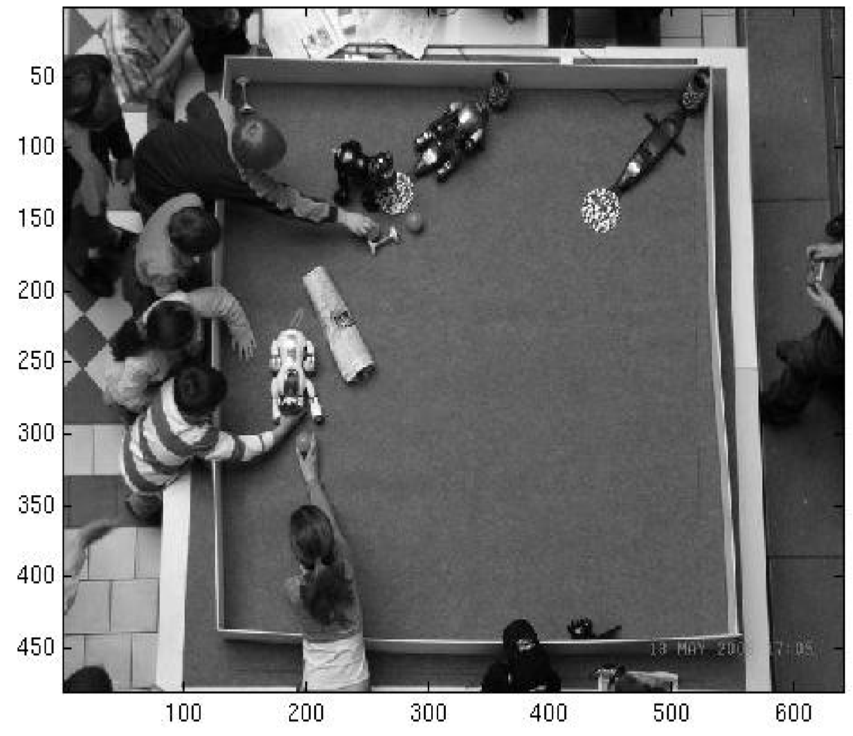


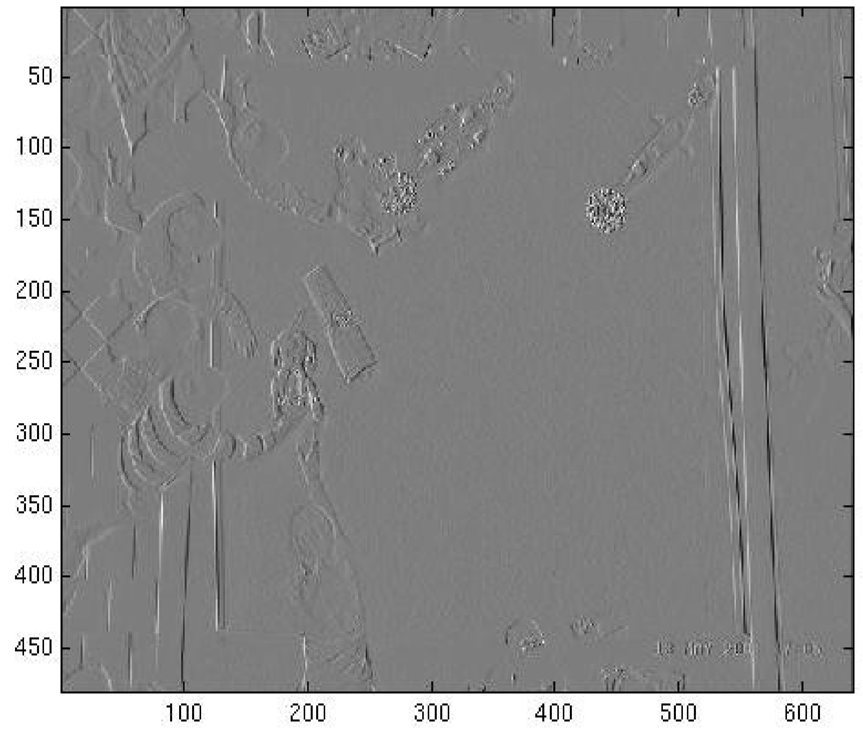
```
[     1 1 1;
      1 1 1;
      1 1 1     ];
```

Matlab code

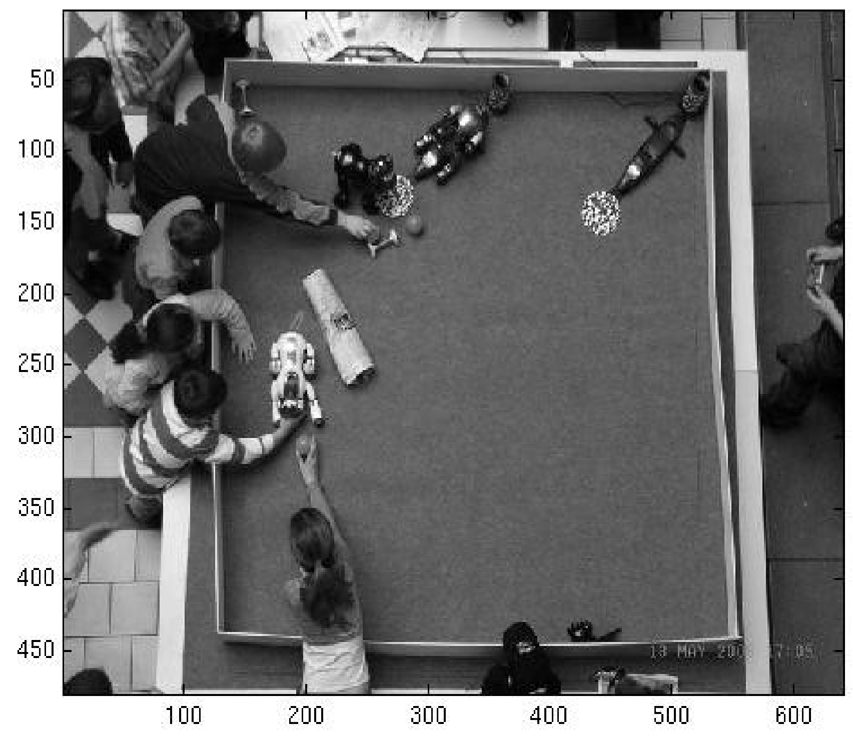
```
% Image processing Beispiele
% Verena V. Hafner, 05/2006

Image=imread('arena.jpg');
colormap(gray);
im=single(Image);
imagesc(Image);

% difference of Gaussian approximation
mask = [-1 -1 -1;-1 8 -1;-1 -1 -1];
test_im1=conv2(im,mask,'same');
figure(2);
colormap(gray);
imagesc(test_im1);

% smoothing
mask = [1 1 1;1 1 1;1 1 1];
test_im5=conv2(im,mask,'same');
test_im5=conv2(test_im5,mask,'same');
test_im5=conv2(test_im5,mask,'same');
figure(6);
colormap(gray);
imagesc(test_im5);

%horizontal edge detection
mask = [1; -1];
test_im3=conv2(im,mask,'same');
figure(4);
colormap(gray);
imagesc(test_im3);

% canny edge filter
test_im4 = test_im2.^2 + test_im3.^2;
figure(5);
colormap(gray);
imagesc(test_im4);

%test_im5 = (test_im4 > 0);
%figure(6);
%colormap(gray);
%imagesc(test_im5);
```

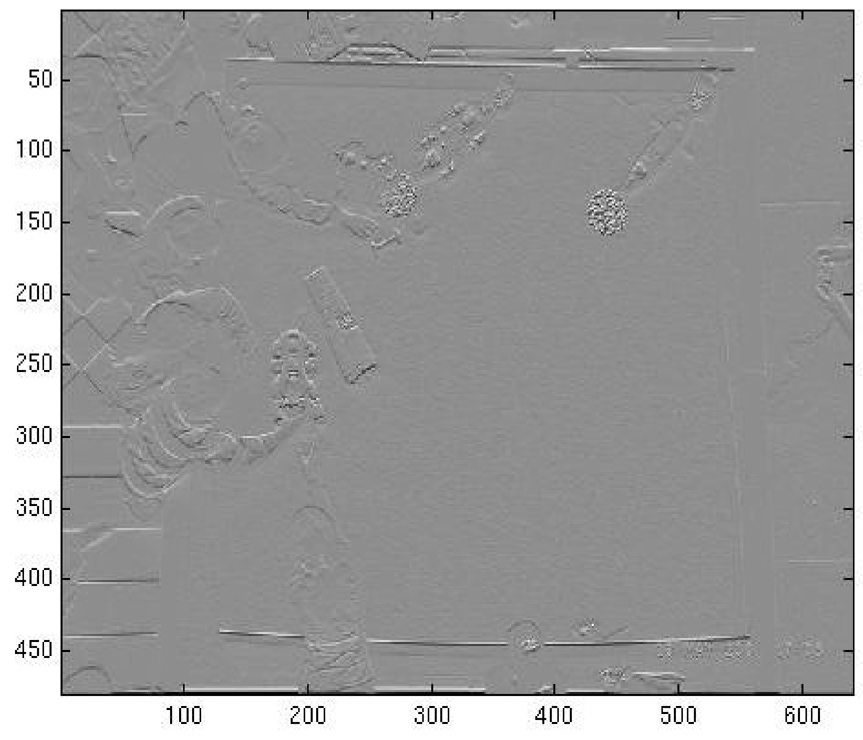
Vertikale Kantenerkennung



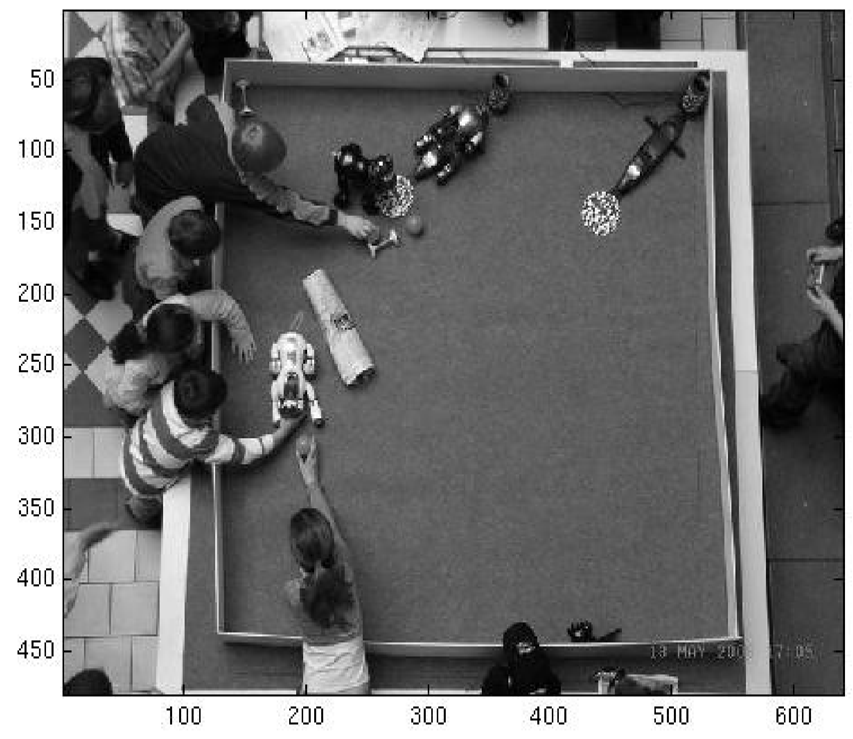


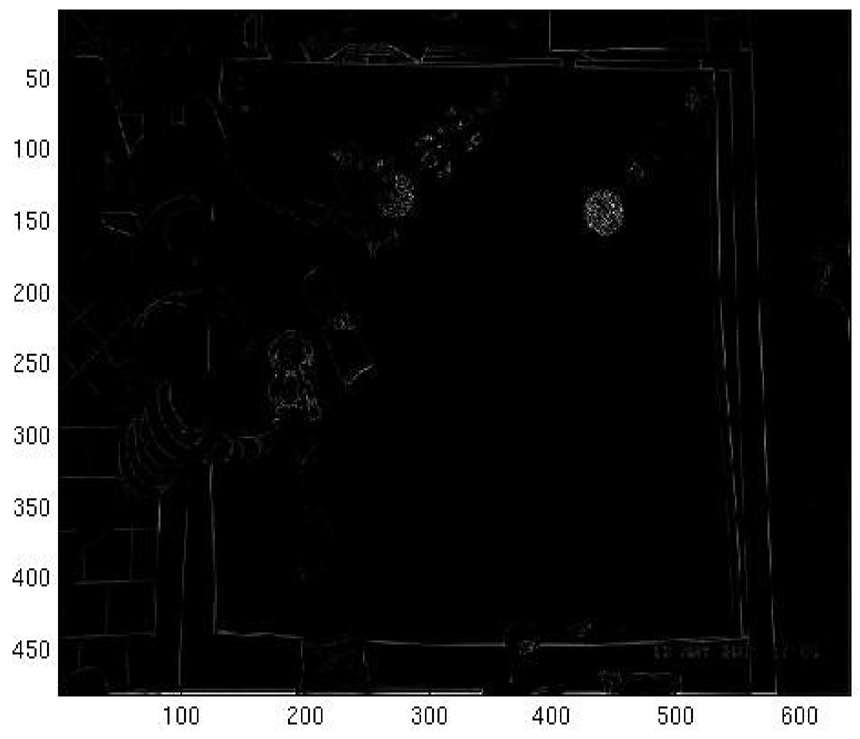
Horizontale Kantenerkennung



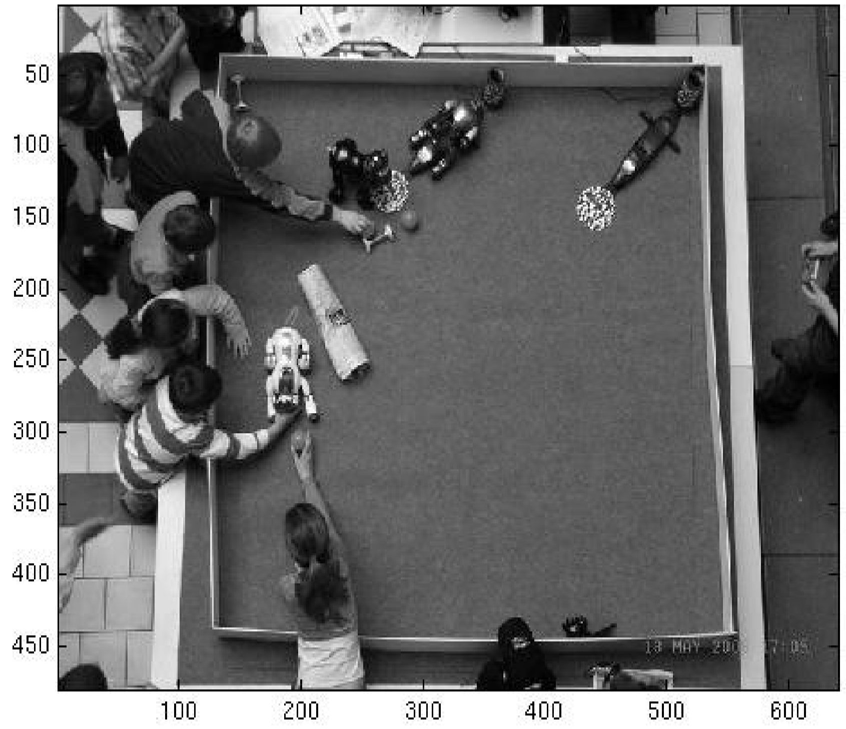


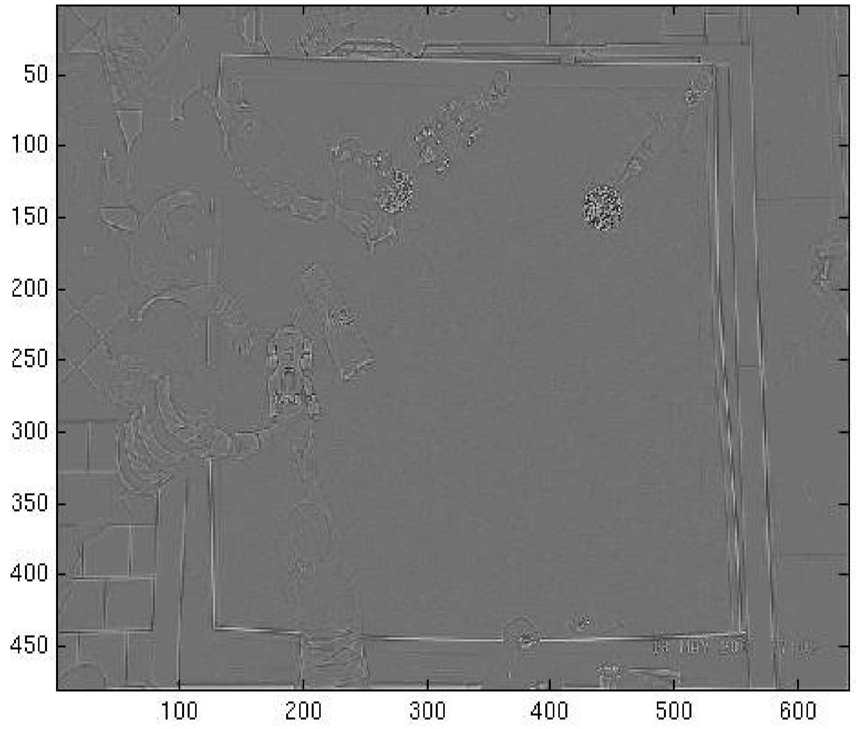
Canny Edge Detector





DoG (Difference of Gaussian) Edge Detektor

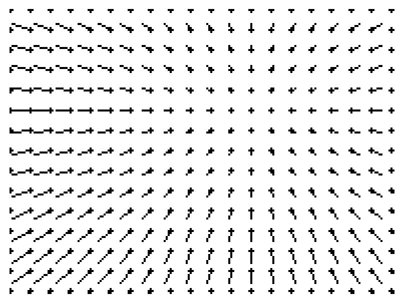
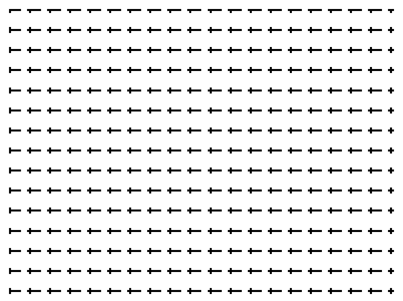
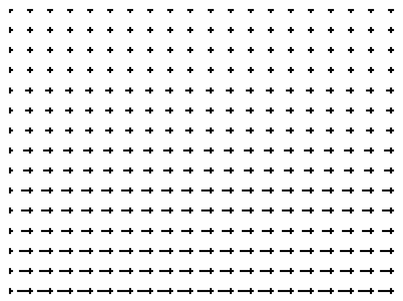




```
[       -1 -1 -1;
        -1  8 -1;
        -1 -1 -1    ];
```

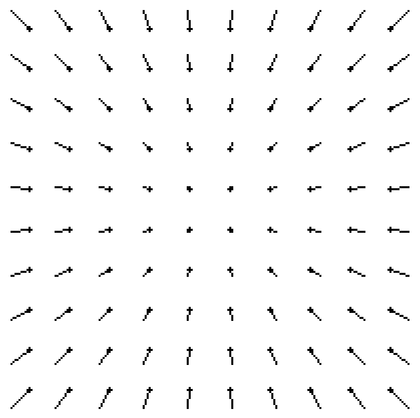
Motion

Optical Flow

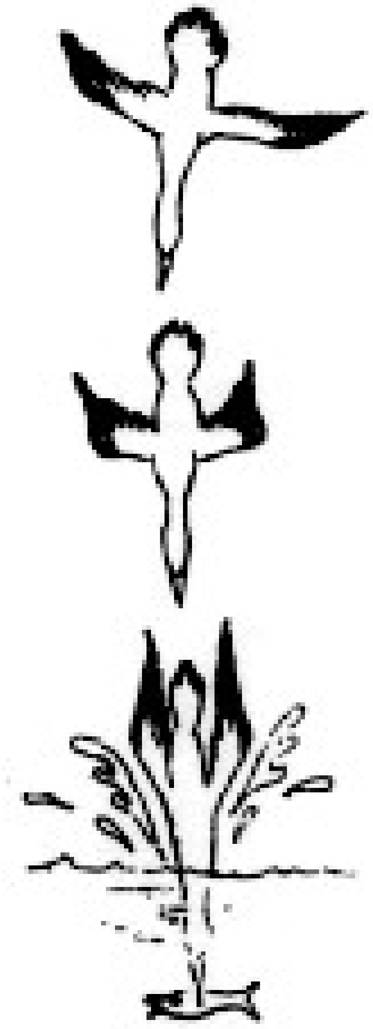
Measuring optic flow using spatial and temporal gradients

The Horn-Schunck method makes use of the simplest local structure that can give information about motion: the grey-level gradient.



TTC (time to contact)

- Diving Gannett’s
  - Birds must draw in wings before contact with water
    - Velocity, height varies
    - TTC constant



Was ist die TTC?

- TTC = size / change in size over time

$$(x) = x / \dot{x}$$

Force

In the Aibo version, you control the joint positions, but I don't want to control my robot joints in position but in force instead. Can you do that with URBI ?

Yes. The Aibo server do not provide a direct way of controlling the joint force, because the underlying hardware is working with positions. For that reason, the URBI server for Aibo doesn't give you a direct access to force control. But if your robot can handle force controlled joints, you can imagine to use joint.force variables instead of joint.val and this will work if the server is properly programmed. It all depends on the underlying hardware.

However, Aibo is equipped with force sensing devices for any joint and you can access them in URBI via the force field of your device: **device.force**

You can also access a calculated force evaluation done by comparing the expected trajectory of the joint with the actual trajectory. When a resisting obstacle is put in the way, this quantity increases as the force. You can access this quantity with the 'e extension on the device position value, like that: **device.val'e**. This mechanism is very general and is available even on robots whose hardware does not provide a force feedback like device.force