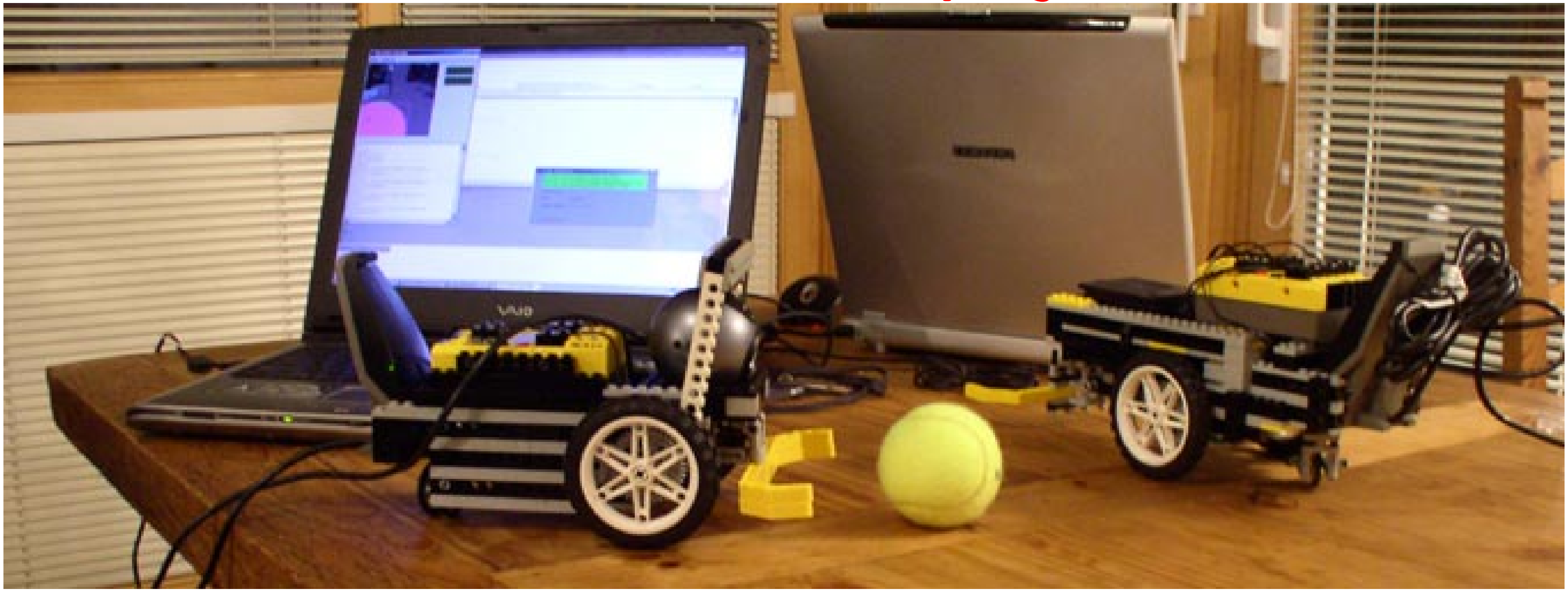
The *SoccerBot* project



Georg Klima, Krystian Szczurek, Peter Wild

January 2006

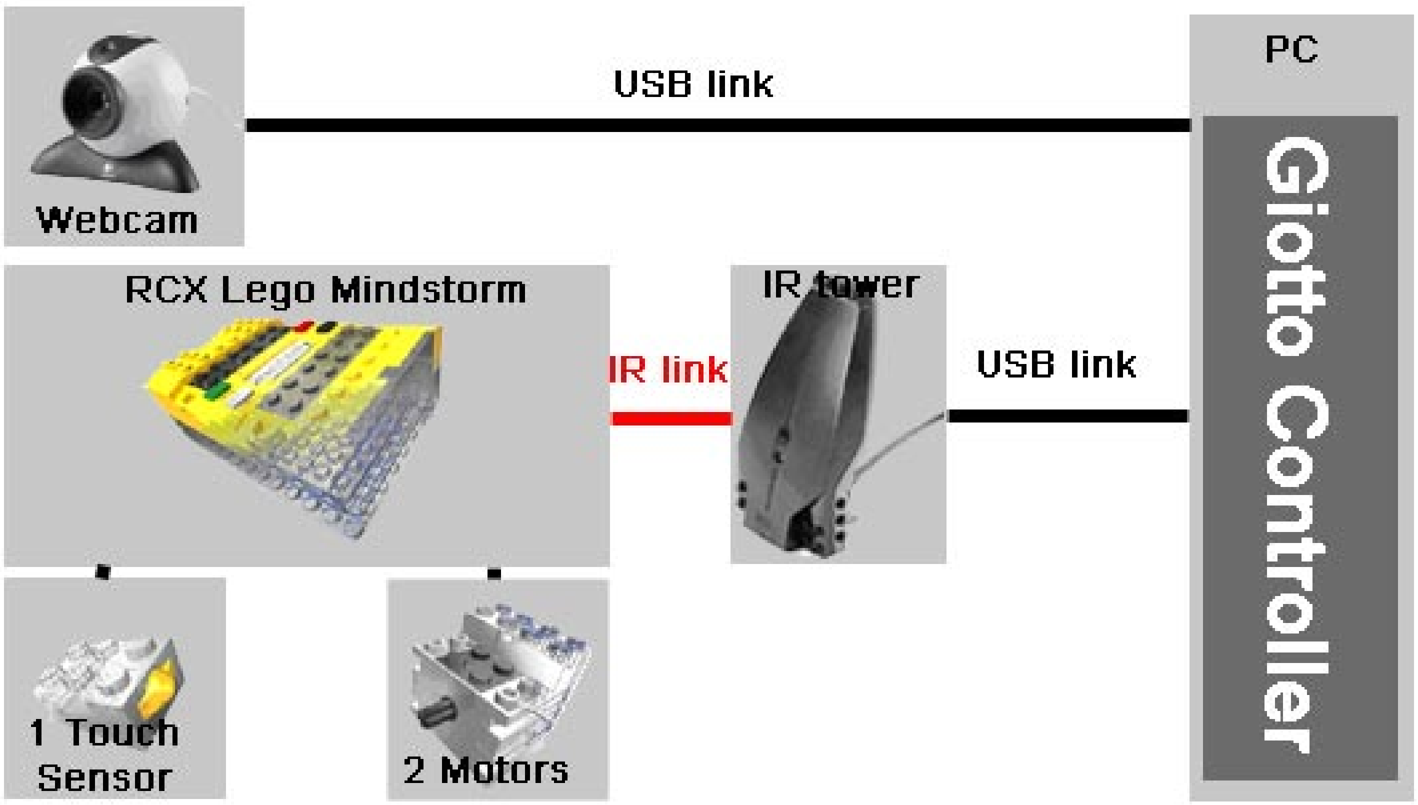
Overview

- **Introduction**: Problem definition, Architectural overview.
- **PC-RCX Communication**: RCXDrive, Communication framework.
- **Object recognition task**: Image segmentation, Split-and-Merge, Object recognition.
- **Controller task**: Finite state automaton, Navigation controller.
- **Future prospects**: Integration Split-and-Merge/Hough Transformation.

Problem definition

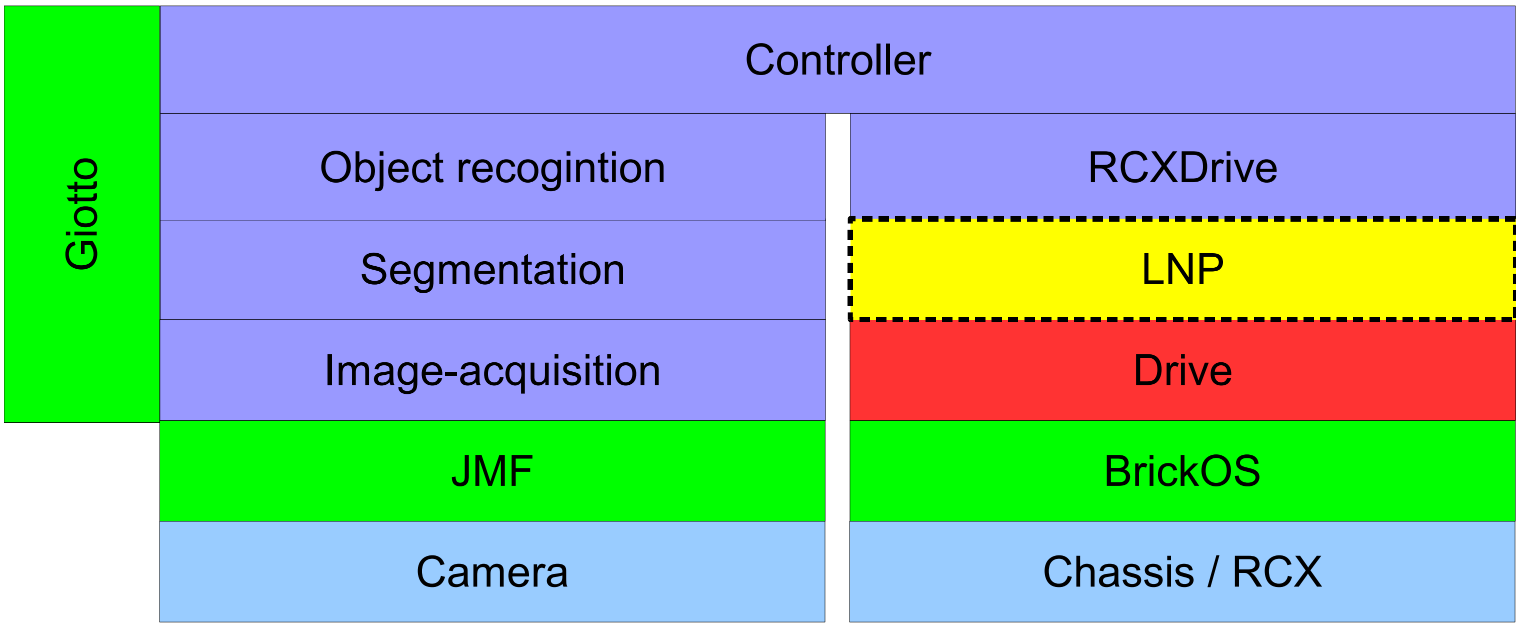
- **Goal**: Develop a *camera-vision equipped* robot with a chassis based on Lego Mindstorms (RCX) featuring *real-time object recognition*.
- **Motivation**: Construct a *semi-autonomous* robot able to recognize and catch a ball, distinguishable from the environment by its color.
- **Requirements**:
  - ⋆ **Real-time**: Reaction time ≤ 300 ms, image processing rate ≥ 10 fps.
  - ⋆ **Quality**: False classifications $< 1\%$, robust against changes in lighting.

Original concept



Architectural overview

Target platform *on PC*: Java / Giotto for timing code, *on RCX*: BrickOS/C.

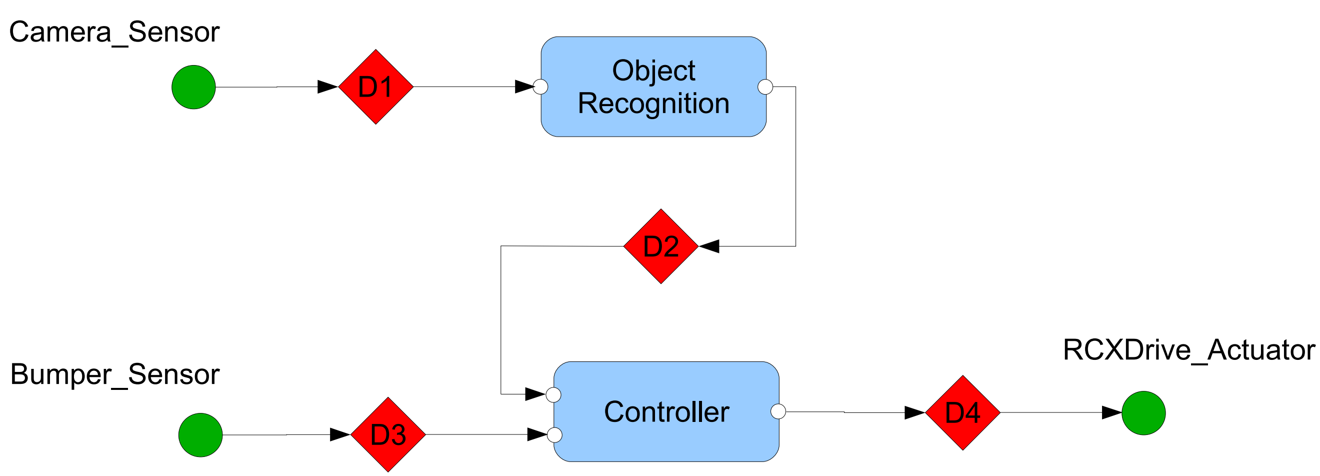


Giotto model (1)

- *Sensors*: `Camera_Sensor`, `Bumper_Sensor`.
- *Actuators*: `RCXDrive_Actuator`.
- **Modes**: Init (Hardware setup), Main (Find-and-catch), DeInit (Shutdown).
- **Mode main** (period 100):
  - ⋆ `ObjectRecognition_Task` (freq 1):
    acquires image and performs segmentation + object recognition.
  - ⋆ `Controller_Task` (freq 1):
    navigation based on internal state, object recognition and bumper sensor.

Giotto model (2)

```
mode main() period 100 {
    actfreq 1 do RCXDrive(RCXDrive_Driver);
    exitfreq 1 do deinit(ExitMain_Driver);
    taskfreq 1 do ObjectRecognition(ObjectRecognition_Driver);
    taskfreq 1 do Controller(Controller_Driver);
}
```



PC-RCX Communication

- Bidirectional communication between RCX and PC transmits: **Drive-commands**, **Bumper-events**.

- Emulated sensor / actuator on PC
  - ⋆ Motors, actuator in Giotto
  - ⋆ Bumper, sensor in Giotto

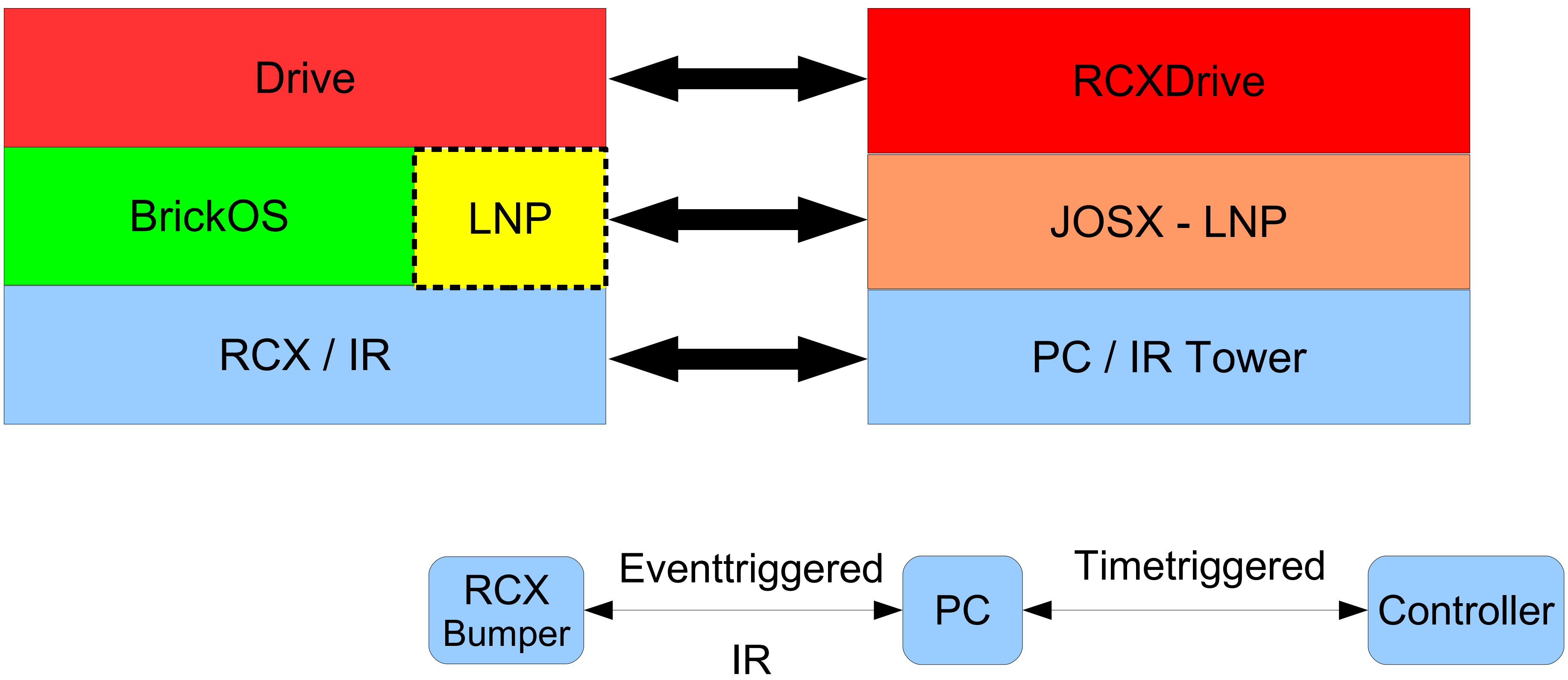
- Communication requirements:
  - ⋆ Reliability: integrity messages.
  - ⋆ Low-latency: < 100 ms communication delay.

RCXDrive - PC part / RCX part

- PC part:
  - Based on a modified JOSX stack.
  - Uses Lego Networking Protocoll packet oriented integrity messages.
  - Java DataInput / DataOutput compatible.

- RCX part:
  - Controller-functions: **Motorcontrol**, **Communication**, **Bumperevents**.
  - Multithreaded
  - Enhancements:
    - Use rotation sensors for straight driving
    - Still some improvements in terms of latency

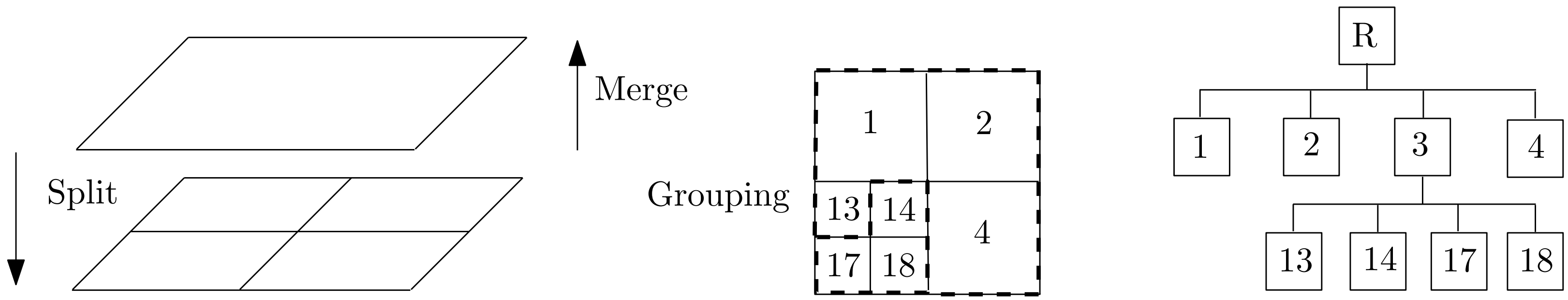
Communication overview



Object recognition task

- **Input port**: 352×288 24-bit still camera image (`BufferedImage_port`).
- **Output port**: best-matching region classifying the ball, if the ball is visible (`Region_port`).
- **Steps**:
  - ⋆ **Segmentation** of the image into regions using *Split-and-Merge* (S&M) algorithm with homogenity criteria in *HSV color space*. (estimated WCET: 80 ms)
  - ⋆ **Object recognition** of ball using closest *color-distance* to *threshold* in *CIE Lab color space* and *aspect ratio* as *shape-test*. (estimated WCET: 10 ms)

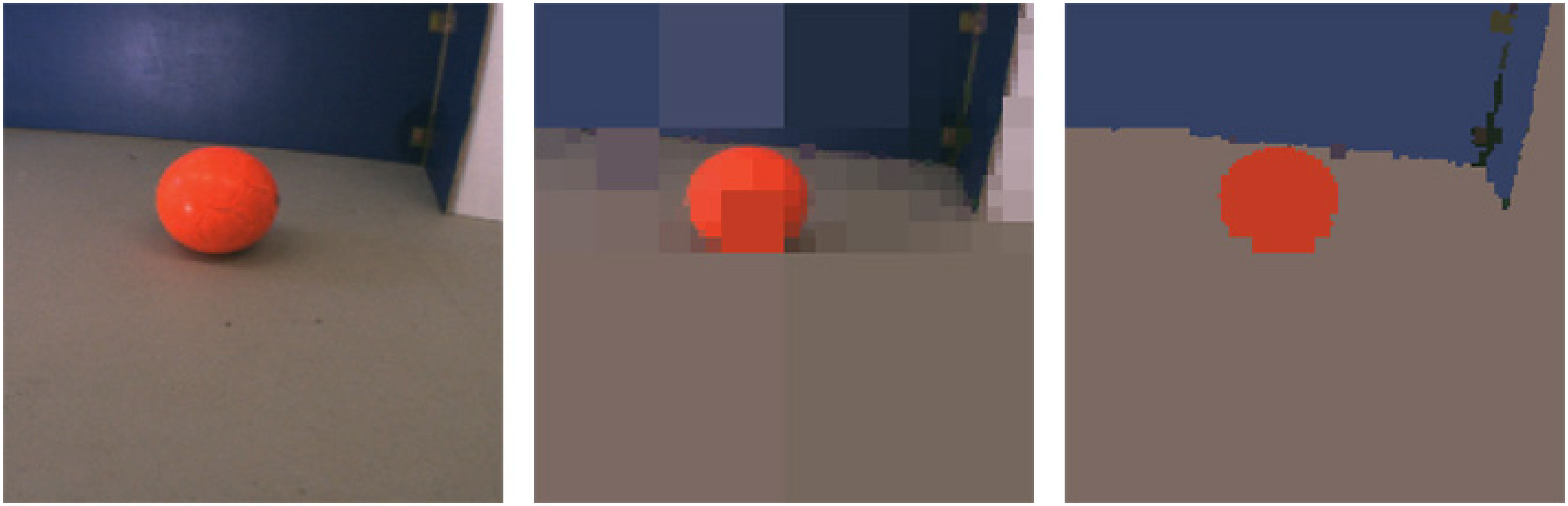
Image Segmentation using Split and Merge



- S&M is a region-based segmentation algorithm by Horowitz et Pavlidis, that divides images into regions and merges *homogeneous* regions recursively, using quadtrees.

- Our slightly modified version consists of 3 steps: **Split**: Initial full segmentation; **Merge**: if 4 homogeneous sub-regions may be merged to a region fulfilling the *homogenity criteria*, they are merged; **Grouping**: if 2 neighboring regions (even at different levels) may be merged to a region fulfilling the *uniformity criteria*, they are merged.

Split and Merge: example



(a) original 256×256 24-bit picture, (b) result after Merge: 748 regions, (c) result after Grouping: 19 regions.

Split and Merge: homogenity criteria (1)

- **Possible criteria**: *Min-Max-Difference* ($H(r) = true \Leftrightarrow max_r - min_r \leq thresh$), *Color-Variance* ($H(r) = true \Leftrightarrow \sigma_r^2 + \sigma_g^2 + \sigma_b^2 \leq thresh$), etc.
  - ⋆ **Problems**: noise, lighting, few large regions.
- **Used criteria**: checks whether the average colors in *HSV-space* of 2 regions are *similar* according to matrices by Volker Rehrmann, Univ. of. Koblenz:

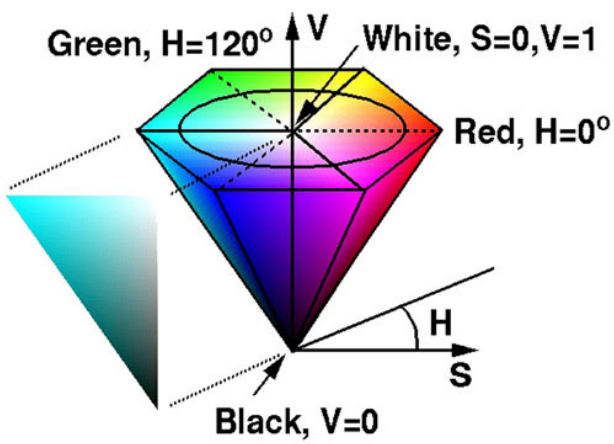
$$D(c_1, c_2) := \begin{cases} true, & for\ c_1\ similar\ to\ c_2 \\ 0 & otherwise. \end{cases}$$

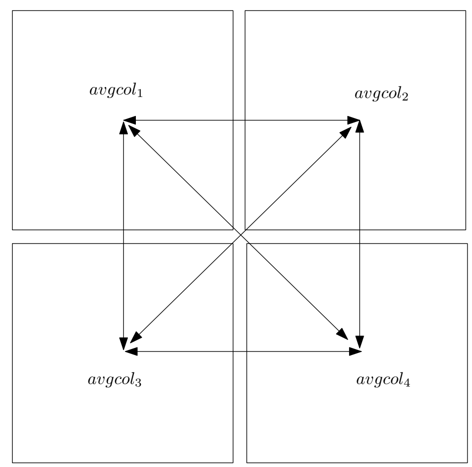
$$D((h_1, s_1, v_1), (h_2, s_2, v_2)) = true \Leftrightarrow |h_1 - h_2| < hue_t \wedge |s_1 - s_2| < sat_t \wedge |v_1 - v_2| < val_t$$

$$(hue_t, sat_t, val_t) = (hue\_tab(min(s_1, s_2), max(v_1, v_2)), sat\_tab(min(s_1, s_2), max(v_1, v_2)), val\_tab(min(s_1, s_2), max(v_1, v_2)).$$

Split and Merge: homogenity criteria (2)

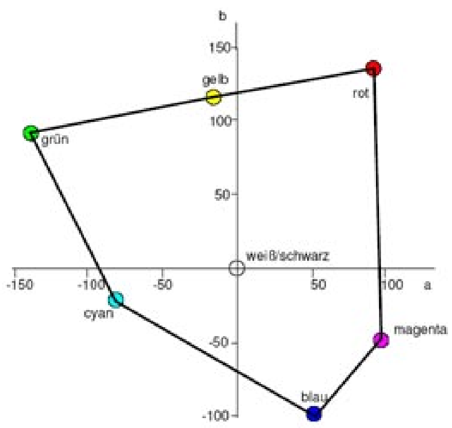
- The homogenity criteria is pairwise applied bottom-up for 4 subregions.
- **Results**: *HSV-space* has better stability against changes in lighting, fast calculation.





Object recognition

- For the detection of the ball, find the region with closest *color-distance* to *threshold* in *CIE Lab color space* with:
  - ⋆ *shape test* is positive: $0.5 \leq height/width \leq 2.5$,
  - ⋆ *color test* is positive: $D(c_{reg}, c_{thres}) = true$.



CIE Lab color space is useful, since euclidian distances between *CIE Lab* colors are near a perceptual measure of color difference.

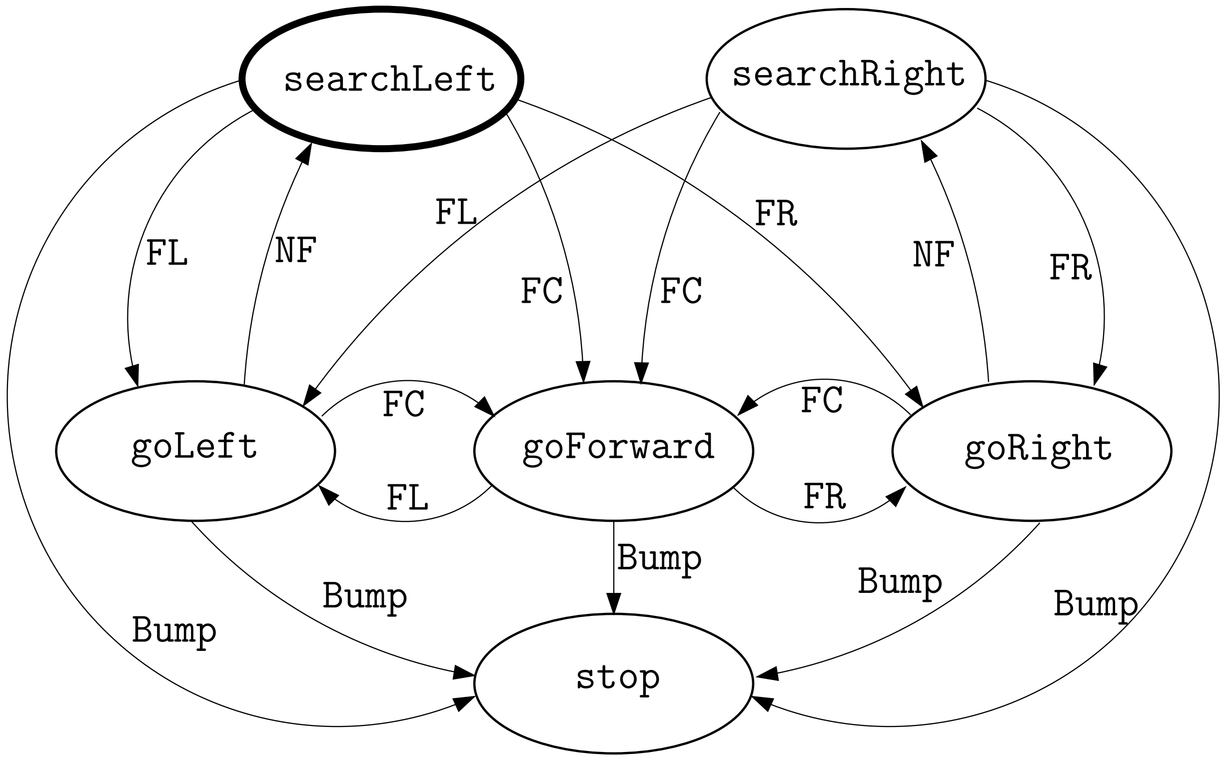
Controller

- Our Controller_Task implements the localization of a tennis ball, moving towards it and stopping after it has been reached.
  - ⋆ Initially the *SoccerBot* rotates left around its axis in order to search for the ball.
  - ⋆ In case the ObjectRecognition_Task returns a valid ball:
    - ∗ Image is split up into three regions: “left”, “center” and “middle”.
    - ∗ According to the region, CMD_CURVE_LEFT, CMD_FORWARD or CMD_CURVE_RIGHT are written to the output port.

- If a ball is not found, then the *SoccerBot* turns either left (CMD_TURN_LEFT) or right (CMD_TURN_RIGHT), according to the last known position of the ball.

- Possible states: FOUND_CENTER, FOUND_LEFT, FOUND_RIGHT, FOUND_CAUGHT, SEARCH_RIGHT_TURN and SEARCH_LEFT_TURN.

Controller: Finite state automaton

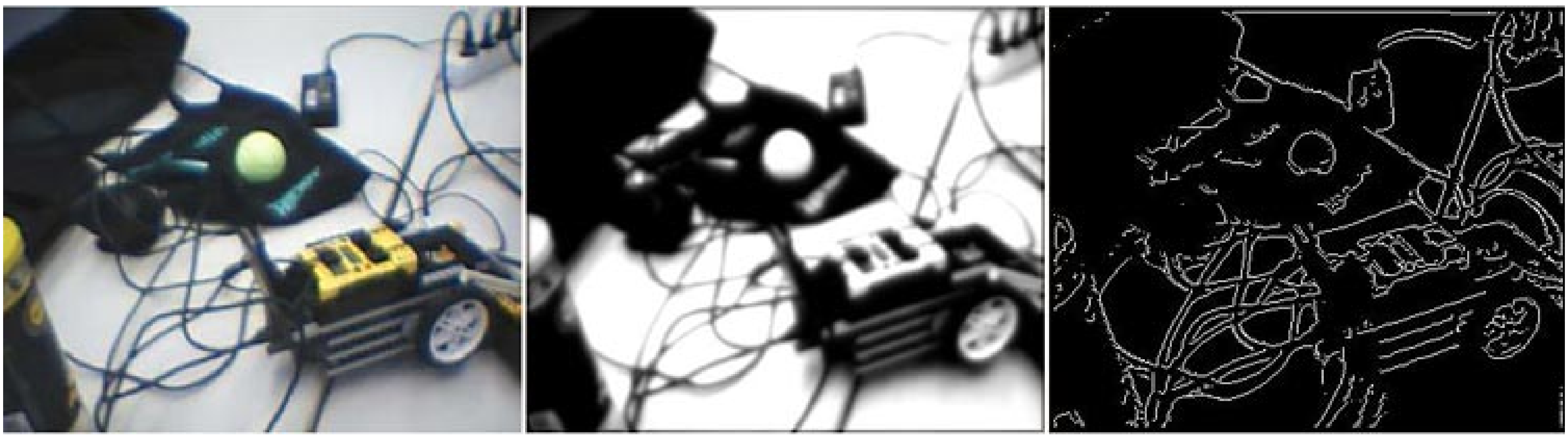


Hough Transformation (1)

- We have also investigated the possibilities of the hough transformation (HT) for our object recognition task.
  - ⋆ The HT enables a search for standard geometric shapes (lines, circles, curves, ...) within images.
  - ⋆ Main advantage: works very well for low quality images and for overlaping objects.
  - ⋆ Main disadvantage: long execution times because of its relatively high complexity:
    - ∗ Line recognition - time & space: $O(n^2)$
    - ∗ Circle recognition - time & space: $O(n^2 * r)$
  - ⋆ One important fact for this presentation is, that it operates on binary images.

Hough Transformation (2)

- The first two steps of the HT:
  1. Color image $\rightarrow$ gray scale image
  2. gray scale image $\rightarrow$ binary edge image



Future Prospects (1)

- The Circular HT (CHT) is relatively slow (about $200 - 300$ ms for a 352×288 RGB image).
  - ⋆ There are still possibilities to speed up CHT.
- CHT can also return a quality-measure of the found circle
  - ⋆ A hybrid method using S&M and HT could be used: S&M could pass parts of the image to the CHT and let it calculate the quality of the circle!

Future Prospects (2)

- At a higher level you could take the image, which S&M generates and pass it on to the CHT or even let S&M already return a binary edge image.
- In order to save calculation time, the CHT is not invoked for all possible circles:
  - ⋆ The predefined radius range could be adopted at runtime.

SoccerBot Demonstration

Thank you for your attention!