Behaving in Space

How Robots Play Soccer

Thomas Röfer

Bremen Institute of Safe Systems

Center for Computing Technology (TZI)

Universität Bremen

Outline of the Tutorial

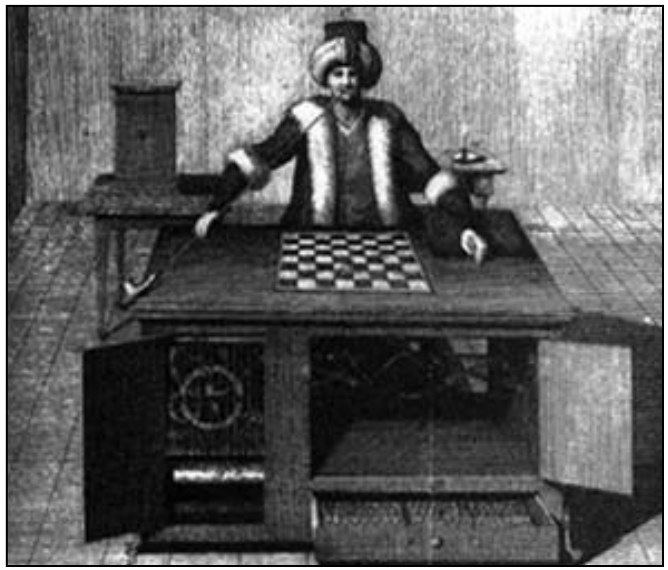
- RoboCup
  - Sony Four-Legged Robot League
  - GermanTeam
- Architecture
  - Processes and process layouts
  - Modules and Solutions
- Perception
  - Systems of coordinates
  - Blob-based Vision
  - Grid-based Vision
- World-Modeling
- Behavior Control
- Motion

Chess – Benchmark for AI

- 1968 Levy bet: “For the next 10 years no chess computer will win against me.”
- 1997: Kasparov against Deep Blue: “I will win and preserve the honor of the human race.”

3½ : 2½

(for Deep Blue)



Why robot soccer?

- 1997 AI Conference in Japan:
  - Realization: Logical problems are quite easy to be solved by high computing power
  - Chess not interesting any more as a benchmark for Artificial Intelligence

  → New Challenge: Development of autonomous robots

Why robot soccer?

- Chess
  - Static environment
  - Discrete data
  - Full information
  - Step based course
  - Logical
  - Single world model
- Robot soccer
  - Dynamic environment
  - Uncertain data
  - Partial Information
  - Real time
  - Autonomous agents
  - Each agent has an own world model

RoboCup – The Goal

- By the year 2050,
develop a team of fully autonomous humanoid robots
that can win against the human world soccer champion team. -

RoboCup – WM 2002 in Fukuoka

Sony Four-Legged Robot League

Sony Four-Legged Robot League

- Sony Aibo ERS-210A
- Four-legged robot
- 20 degrees of freedom
- 400 MHz Mips Processor
- CCD Camera (176x144)
- PSD infrared sensor
- Acceleration sensors
- Microphones & Speaker
- PCMCIA Slot for Wireless-Lan card

The Field

- Size: 2,70m x 4,20m
- Landmarks and goals
- 4 robots per team

The GermanTeam in the Sony Four-Legged Robot League

- Robocup World Championchip
  - Once a year (2003 in Padua, Italy)
  - Participants from all over the world
  - GermanTeam plays as a single united team
- German Open
  - Once a year before the Robocup
  - Open for teams from all over the world
  - German university teams play separately

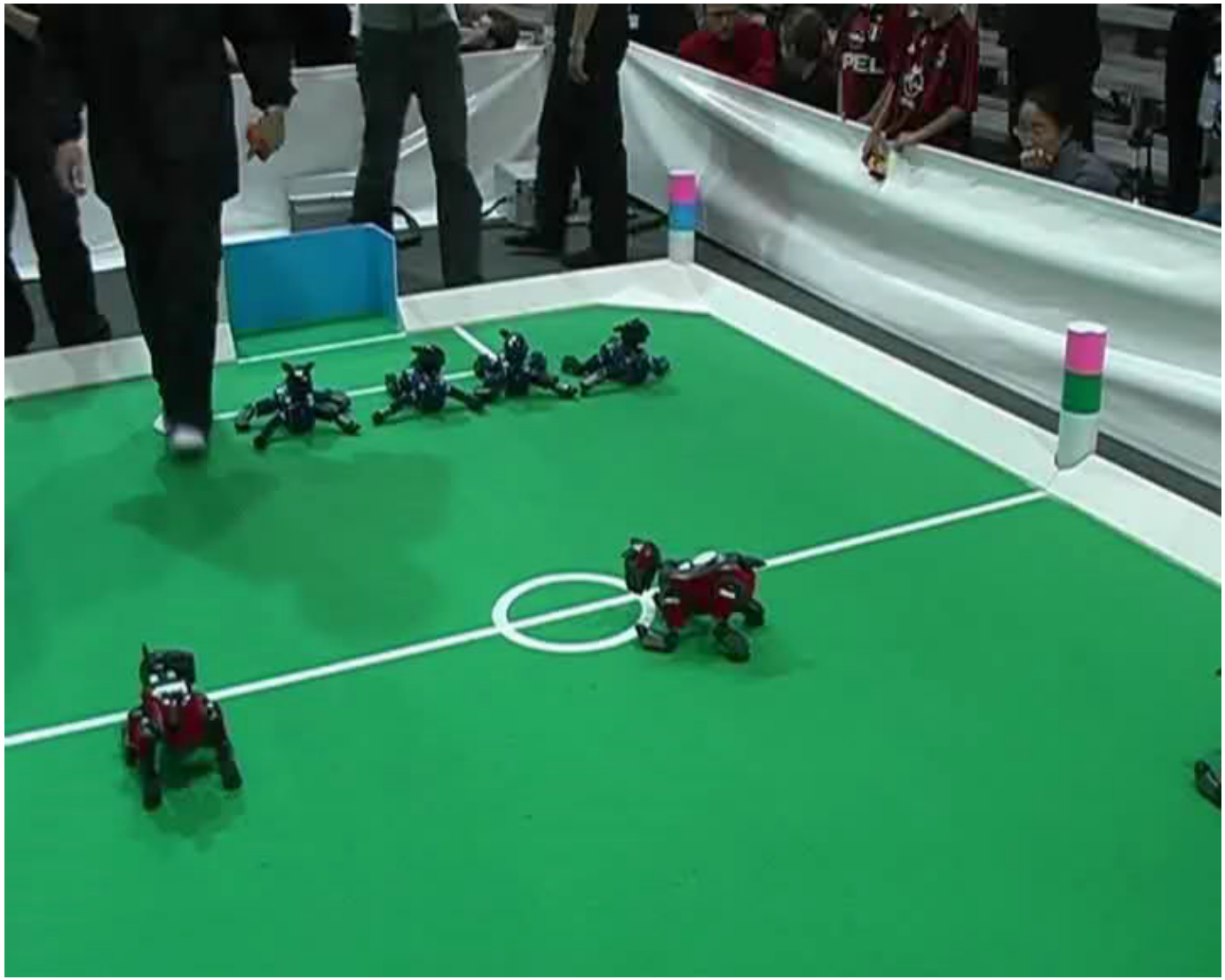
Who is the GermanTeam?

A part of the GermanTeam

The GermanTeam

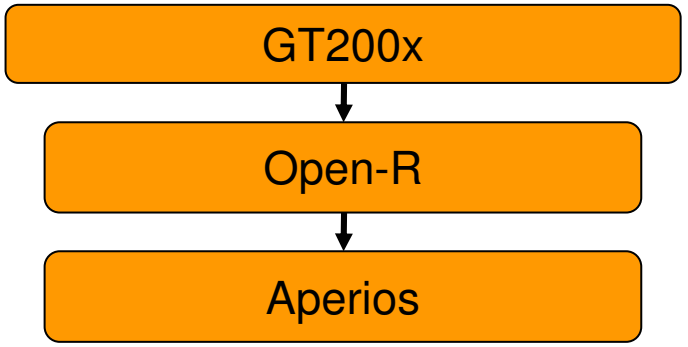
- Community project of four universities:
  - Humboldt Universität zu Berlin
  - Universität Bremen
  - Technische Universität Darmstadt
  - Universität Dortmund
- Communication via:
  - Newsgroup, mailing list, phone
  - Intranet
  - CVS (Concurrent Versions System)
    - *Source code*
    - *Installation files*
    - *Intranet, Papers, Slides (e.g. these ones)*
  - Regular meetings and workshops
- Development platform is Microsoft Developer Studio under Windows 2000/XP

Example: GT2003 vs. UPennalizers, 2nd Half



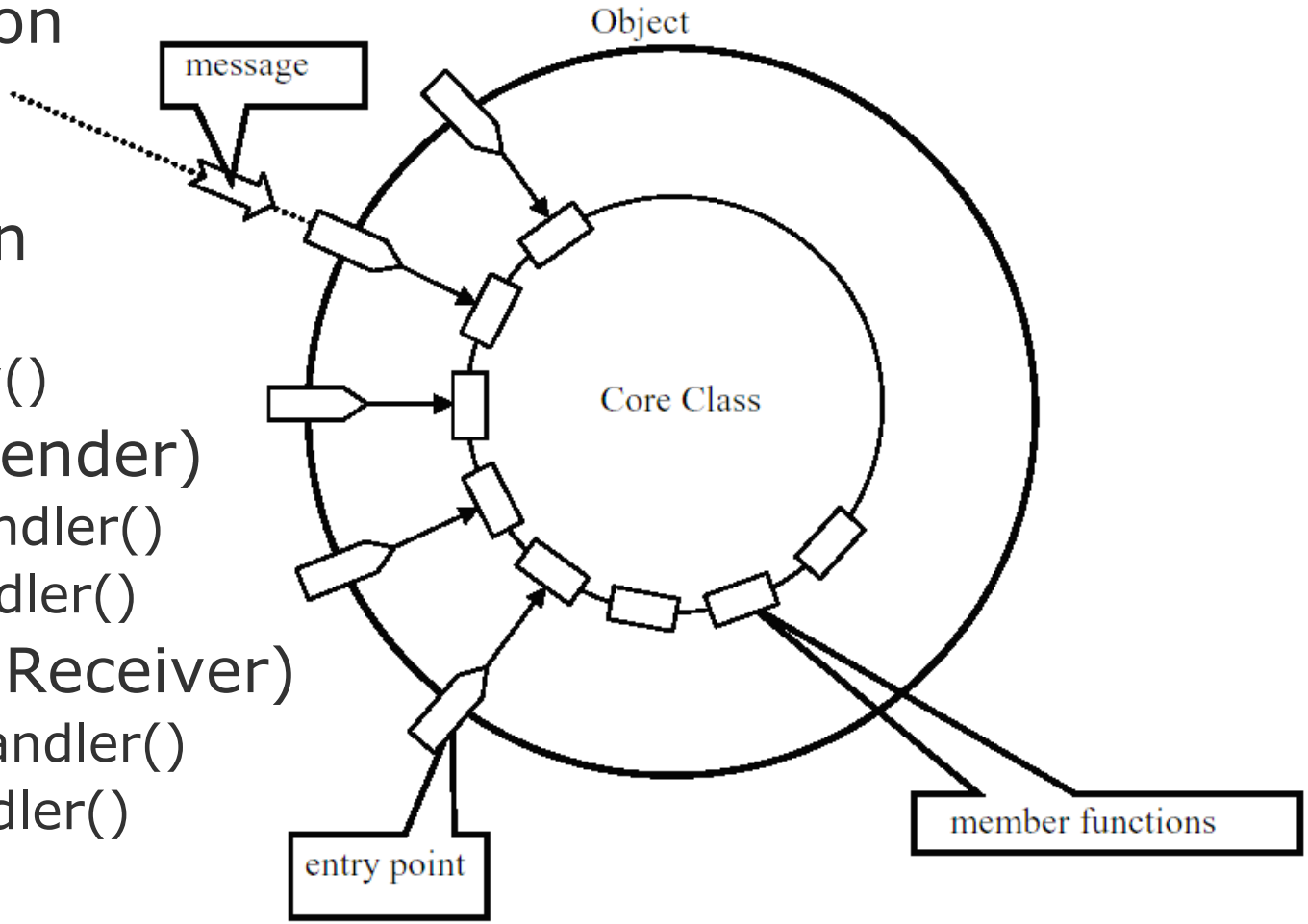
Aibo-OS: Aperios and Open-R

- Everything is written in C++
- Aperios
  - Real-time operating system
  - provides:
    - *Processes*
    - *Inter-process communication*
    - *Memory management*
- Open-R
  - Interface processes to the robot
  - Directly callable functions
- GT200x
  - Only processes



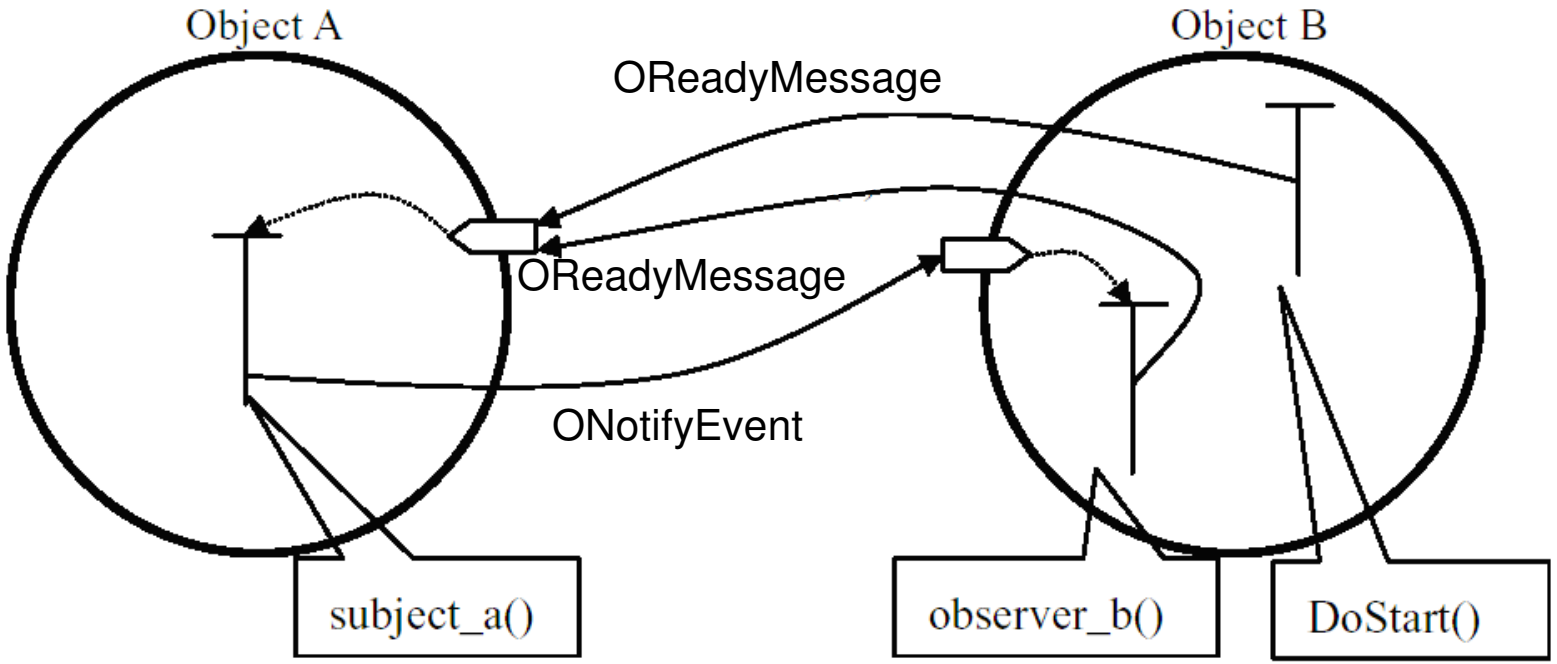
Aperios-Objects (Processes) & Events

- Construction
  - DoInit()
  - DoStart()
- Destruction
  - DoStop()
  - DoDestroy()
- Subject (Sender)
  - ControlHandler()
  - ReadyHandler()
- Observer (Receiver)
  - ConnectHandler()
  - NotifyHandler()
- Timer

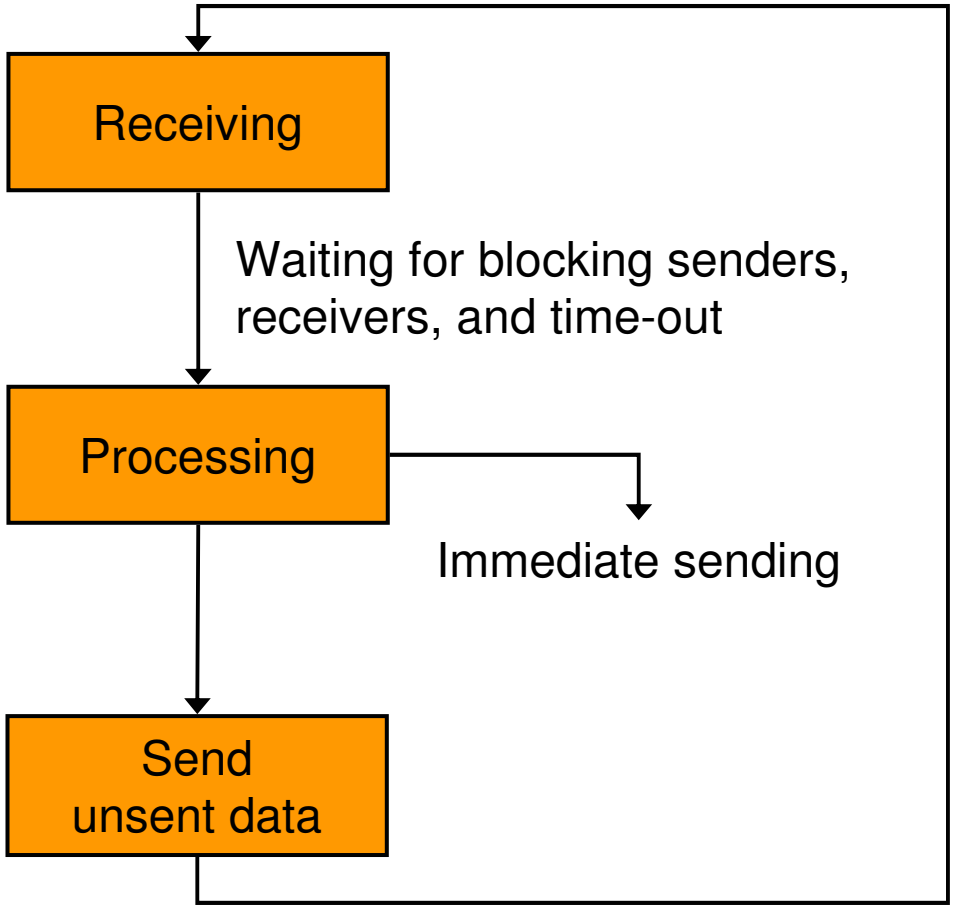


Kommunikation

- DoInit()
  - Establish all communication channels
- DoStart()



GT200x Process Framework



Processes

```
#include "Tools/Process.h"

class Example : public Process
{
  public:
    virtual int main()
    {
      printf("Hello World!\n");
      return 500;
    }
};

MAKE_PROCESS(Example);
```

Packages

```
class NumberPackage
{
  public:
    int number;
    NumberPackage() {number = 0;}
};

Out& operator<<(Out& stream,
                const NumberPackage& package)
{
  return stream << package.number;
}

In& operator>>(In& stream, NumberPackage& package)
{
  return stream >> package.number;
}
```

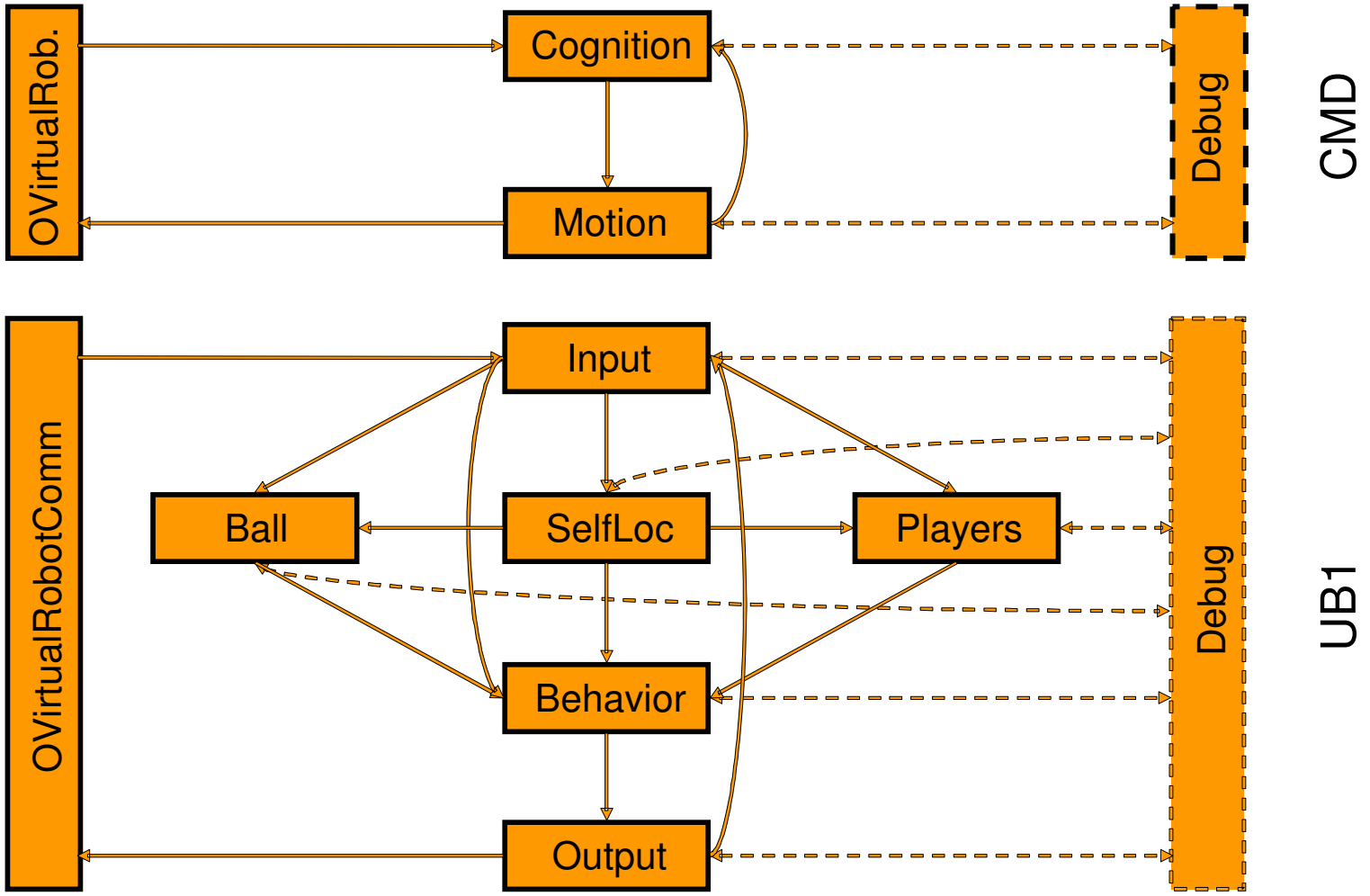
Sender

```
#include "Tools/Process.h"

class Example1 : public Process
{
  private:
    SENDER(NumberPackage);
  public:
    Example1() : INIT_SENDER(NumberPackage,false) {}

    virtual int main()
    {
      ++theNumberPackageSender.number;
      theNumberPackageSender.send();
      return 100;
    }
};

MAKE_PROCESS(Example1);
```

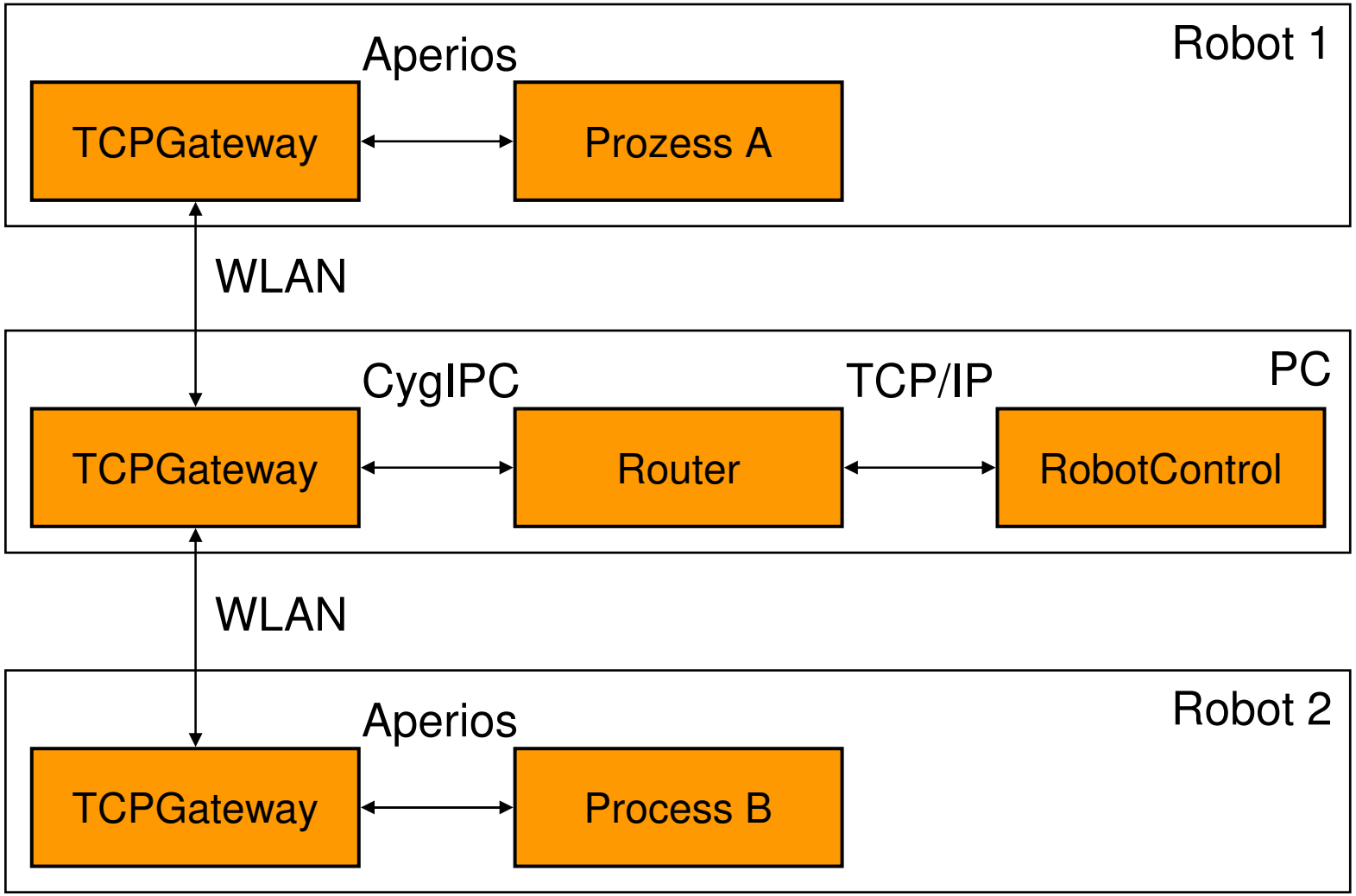
Receiver

```
#include "Tools/Process.h"

class Example2 : public Process
{
  private:
    RECEIVER(NumberPackage);
  public:
    Example2() : INIT_RECEIVER(NumberPackage,true) {}

    virtual int main()
    {
      printf("Number %d\n",
             theNumberPackageReceiver.number);
      return 0;
    }
};

MAKE_PROCESS(Example2);
```

Process Layouts

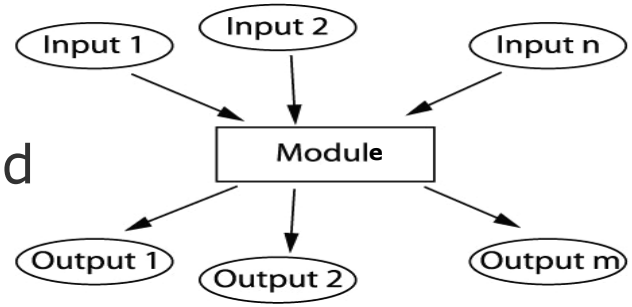


WLAN

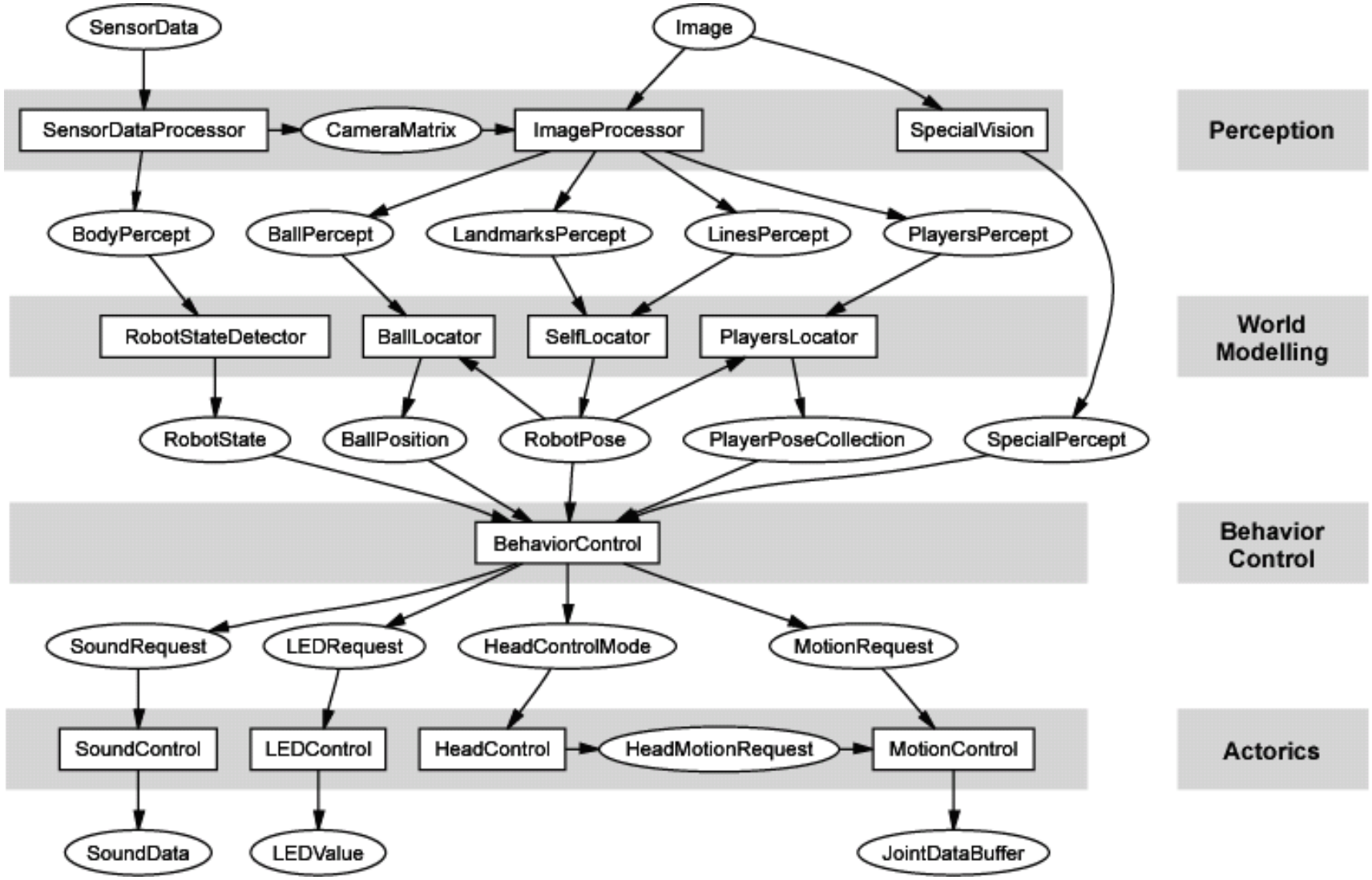

Robot 1
Aperios
TCPGateway
Prozess A
WLAN
PC
CygIPC
TCP/IP
TCPGateway
Router
RobotControl
WLAN
Robot 2
Aperios
TCPGateway
Process B

Modules and Solutions

- Modules
  - Definition of partial tasks in information processing
  - Interaction with other modules / with the robot only via well-defined data structures (intermediate representations)
  - No usage relations between them
  - Sequence of use is defined externally
- Solutions
  - Different solutions for a single module
  - Switching between solutions at runtime
  - Modules can also be switched off

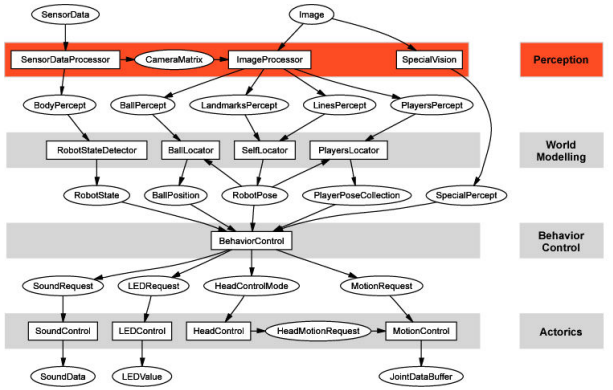


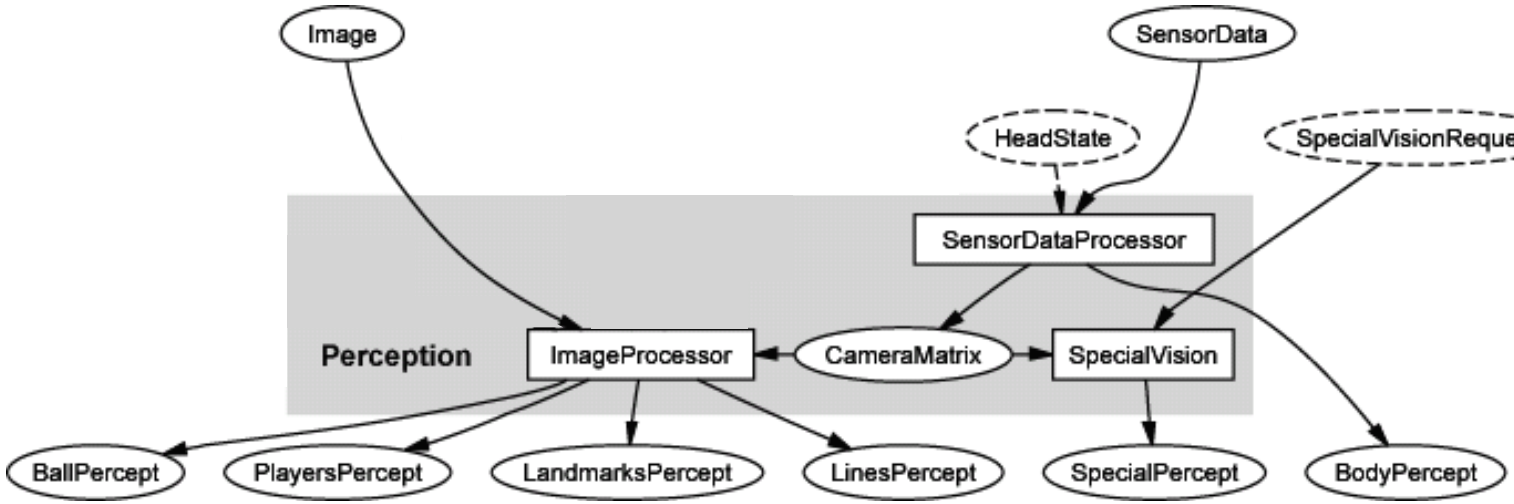
Modules – Overview



Modules – Perception

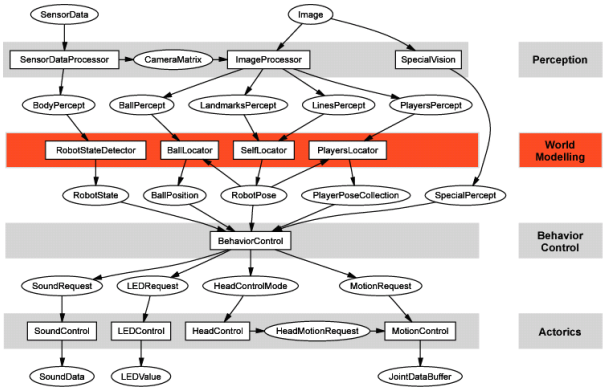
- *SensorDataProcessor*
  - Low pass filtering sensor data
  - Calculation of the *CameraMatrix*
  - Recognition of button presses
- *ImageProcessor*
  - Recognition of relevant objects in an image
- *SpecialVision*
  - Special analyses of an image can be conducted on demand that may take longer

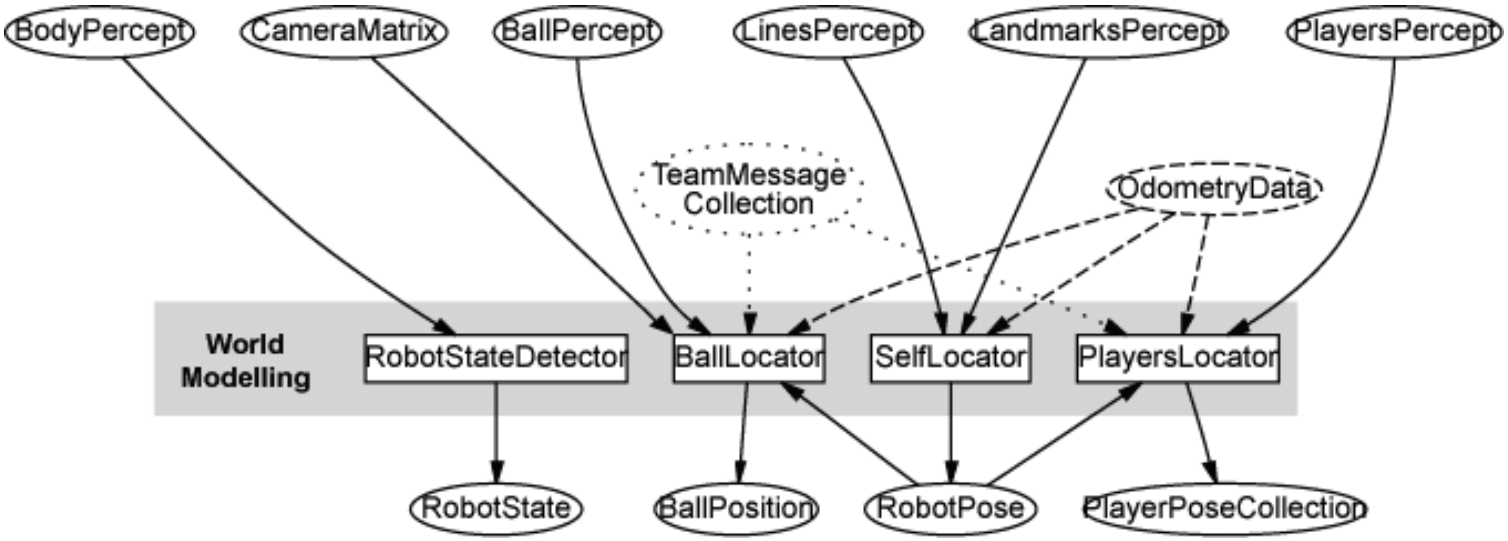




Modules – World Modeling

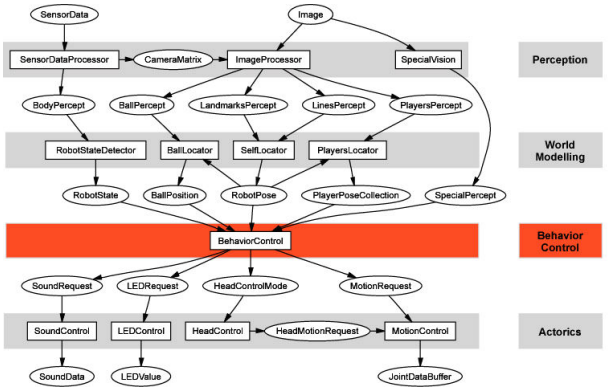
- *SelfLocator, BallLocator, PlayersLocator, ObstaclesLocator, RobotStateDetector*
  - No access to sensor data
  - Integration of different percepts
  - Including information received from teammates
  - Generation of a consistent world model, even about objects not visible in the current image

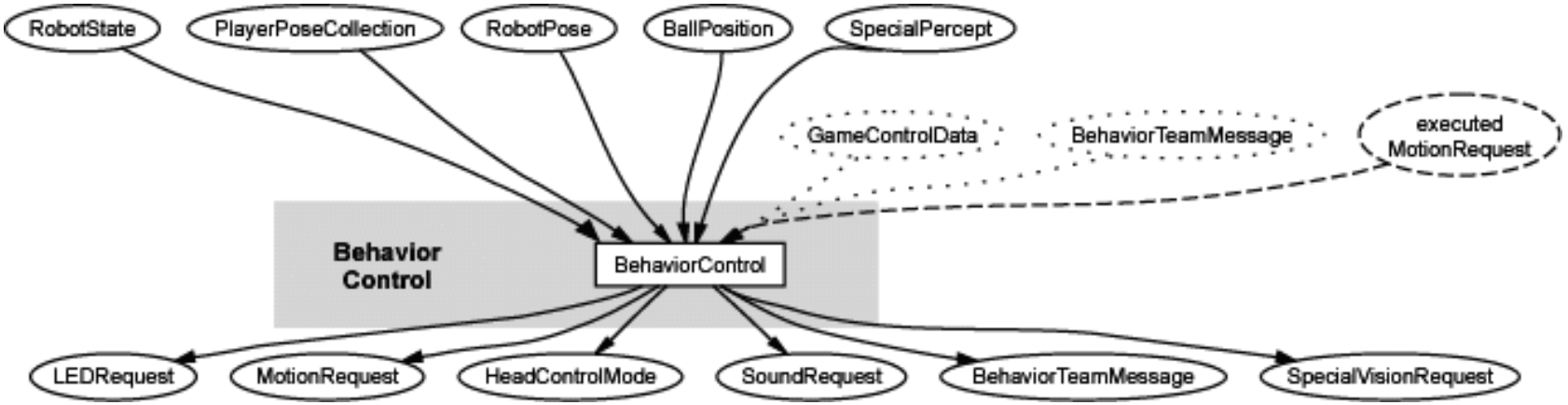




Modules – Behavior Control

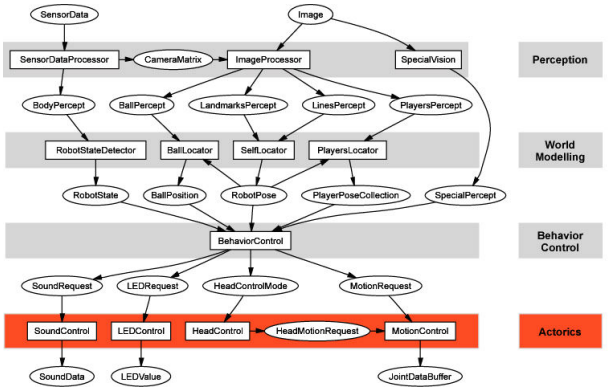
- *BehaviorControl*
  - No access to percepts
  - Commands to actuatorics
  - Requests output via LEDs and loudspeaker
  - Controls robot's attention
  - Influences information processing
  - Communication with teammates
  - Processes commands from the referee

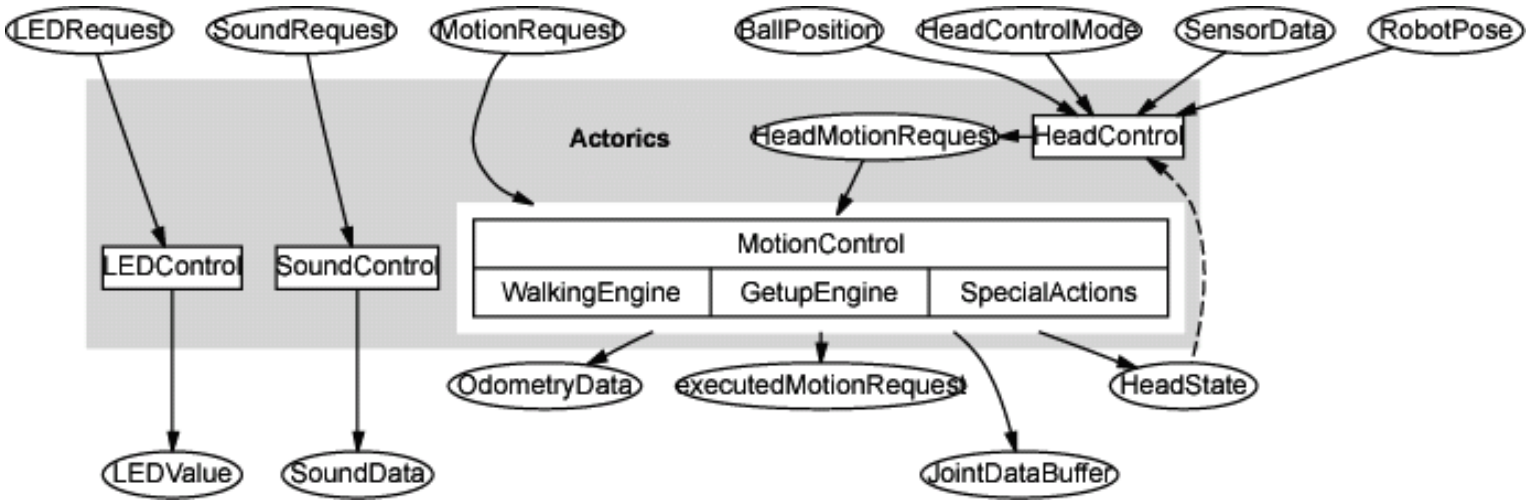




Modules – Actuatorics

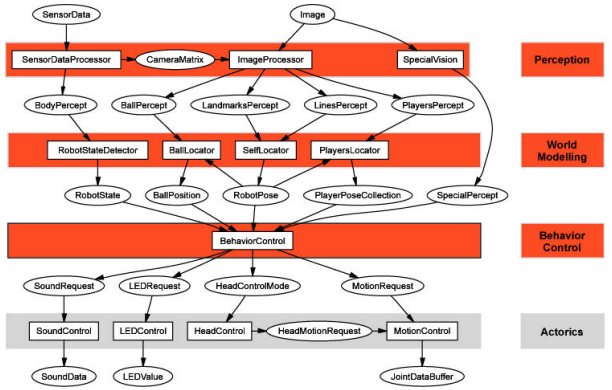
- *HeadControl*
  - View of the camera on the objects of the world
  - Takes preferences of behavior control into account
- *MotionControl*
  - Walking: *WalkingEngine*
  - Kicks, tricks: *SpecialActions*
  - Standing up: *GetupEngine*
  - Integration of desired angles for head joints
- Output for developers: *LEDControl, SoundControl*

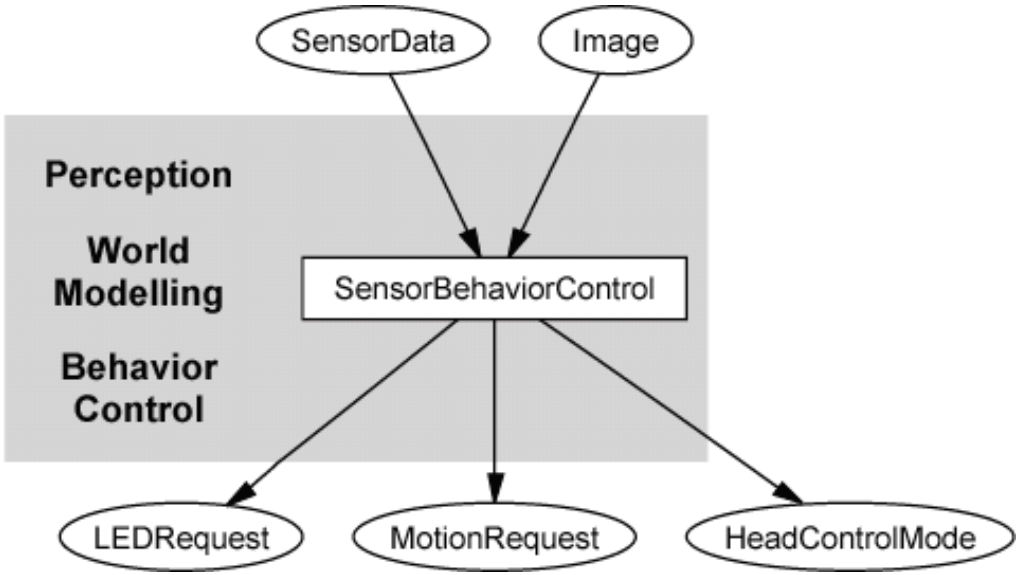




Modules – Sensorimotor Coupling

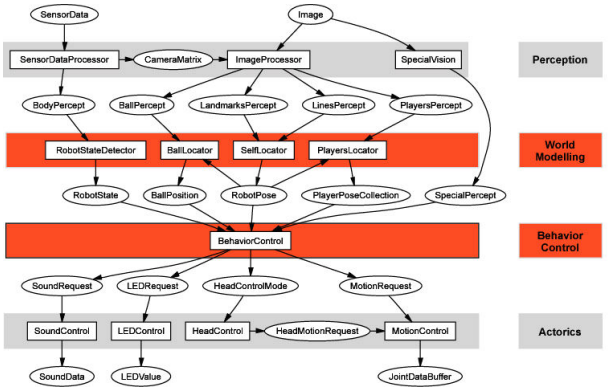
- Direct generation of commands to actuatorics from sensor data: *SensorBehaviorControl*
  - Own image processing
  - Own modeling
  - For experiments

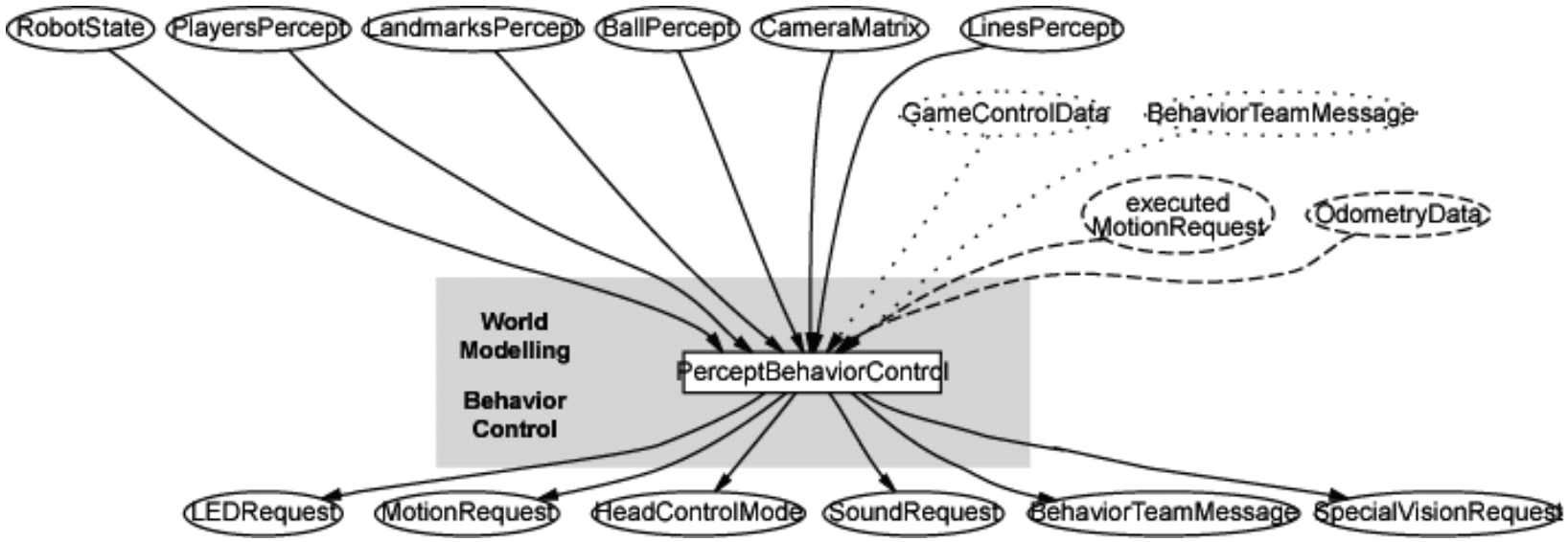




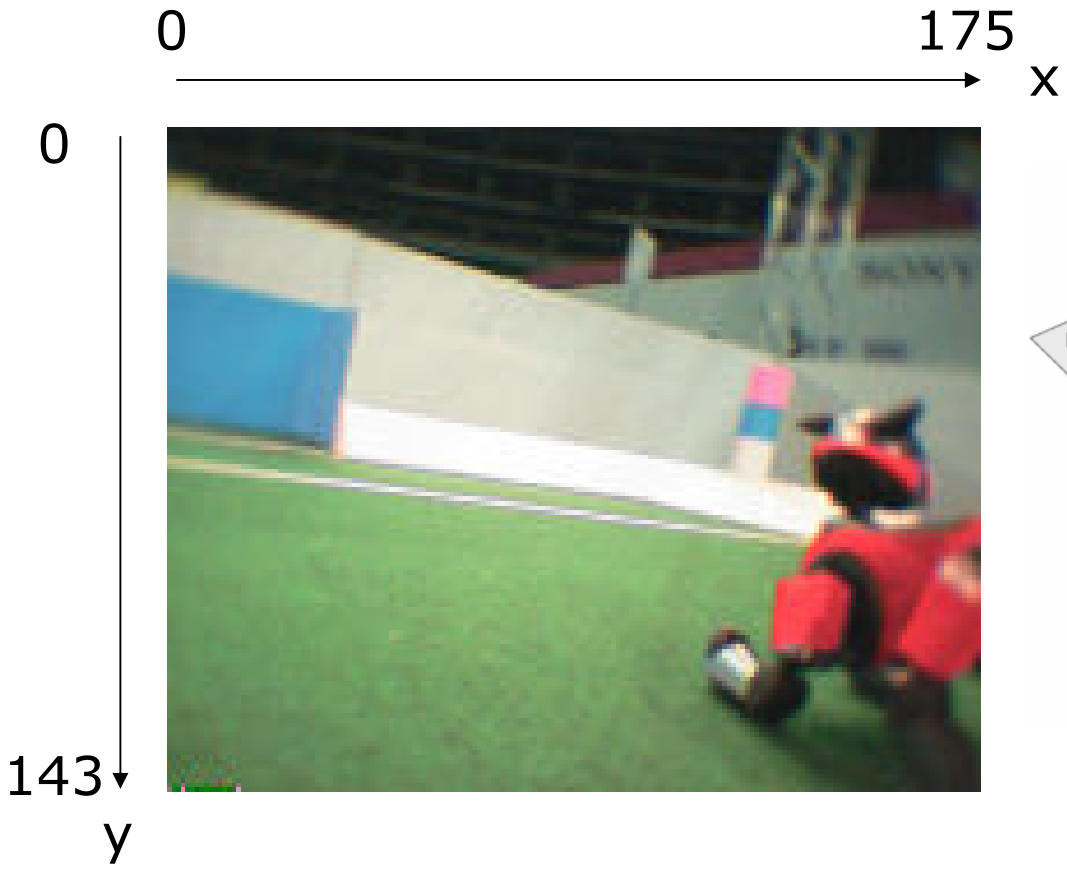
Modules – Percept-Based Control

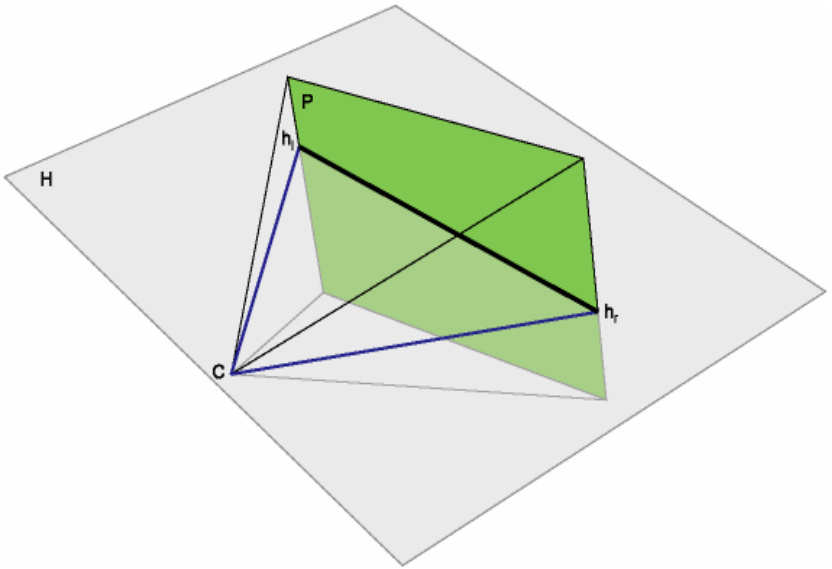
- Direct generation of commands to actuatorics from Percepts: *PerceptBehaviorControl*
  - Own modeling
  - „Probabilistic behavior"





Perception

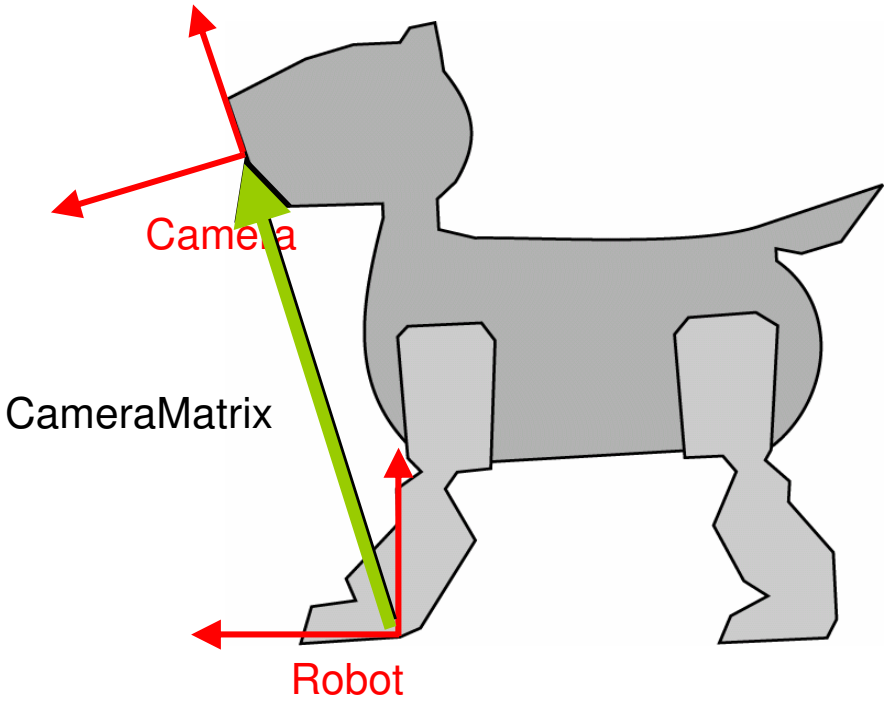


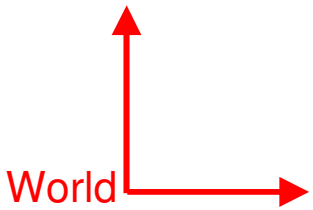


Perception – Position of the Camera (1)

CameraMatrix:

Transformation of the robot's system of coordinates into the system of coordinates of the camera

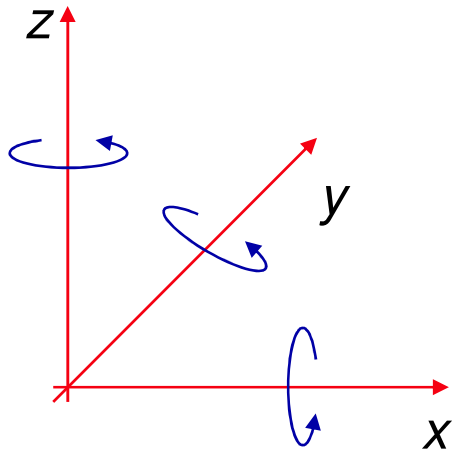




Perception – Position of the Camera (2)

translation by (x,y,z) ↓

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



rotation around z-axis by $\chi \rightarrow$

$$\begin{pmatrix} \cos\chi & -\sin\chi & 0 & 0 \\ \sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

… around y-axis by $\beta \rightarrow$

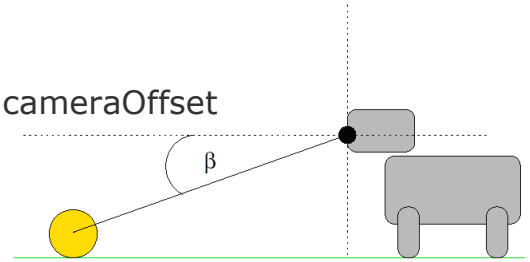
$$\begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

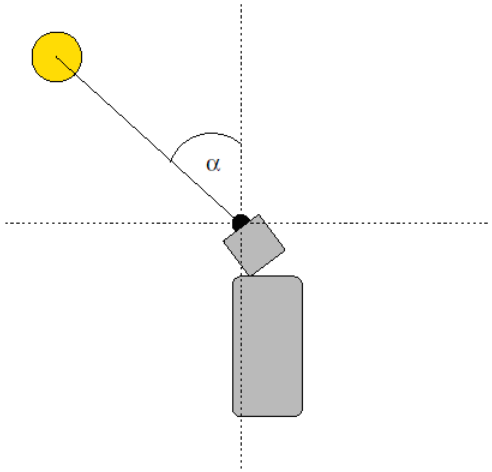
… around x-axis by $\alpha \rightarrow$

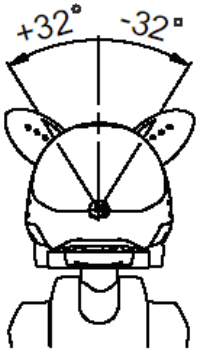
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

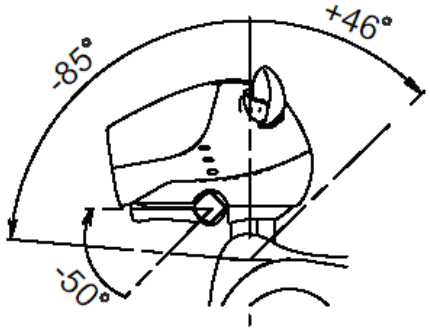
Perception – Position of the Camera (3)

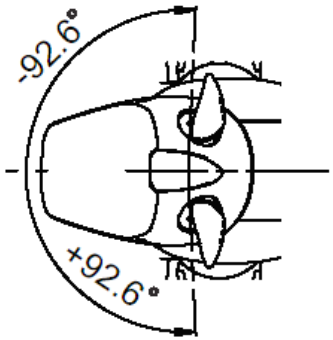
- CameraMatrix
  - $M_{camera} = M_{neckHeight}\ M_{tilt}\ M_{neckLength}\ M_{pan}\ M_{roll}\ M_{cameraOffset}$
- Anglular coordinates
  - $x_{factor} = (\tan(width_{angle} / 2) * 2) / width_{pixel}$
  - $y_{factor} = (\tan(height_{angle} / 2) * 2) / height_{pixel}$
  - $V = M_{camera}.rotation\ (1,\ (width_{pixel} / 2 - x) * x_{factor},\ (height_{pixel} / 2 - y) * y_{Factor})$
  - $\alpha = atan2(V.y, V.x);$
  - $\beta = atan2(V.z, sqrt(V.x^2 + V.y^2));$











Perception – Color Spaces – YUV

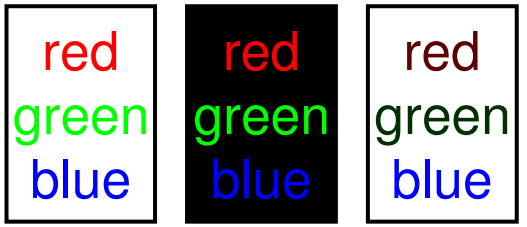
RGB

Y

U

V

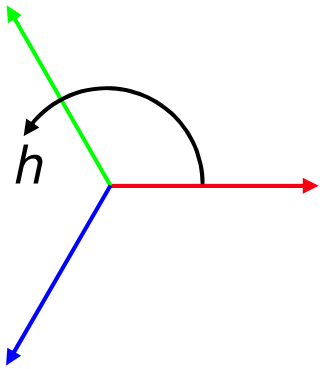
Perception – Color Spaces – Transformations



- Assumption
  - All values are in the range [0..1]
- YUV → RGB
  - $r := \text{fix}(1.164\ (y - 0.0625) + 1.596\ (v - 0.5))$
  - $g := \text{fix}(1.164\ (y - 0.0625) - 0.392\ (u - 0.5) - 0.813\ (v - 0.5))$
  - $b := \text{fix}(1.164\ (y - 0.0625) + 2.017\ (u - 0.5))$

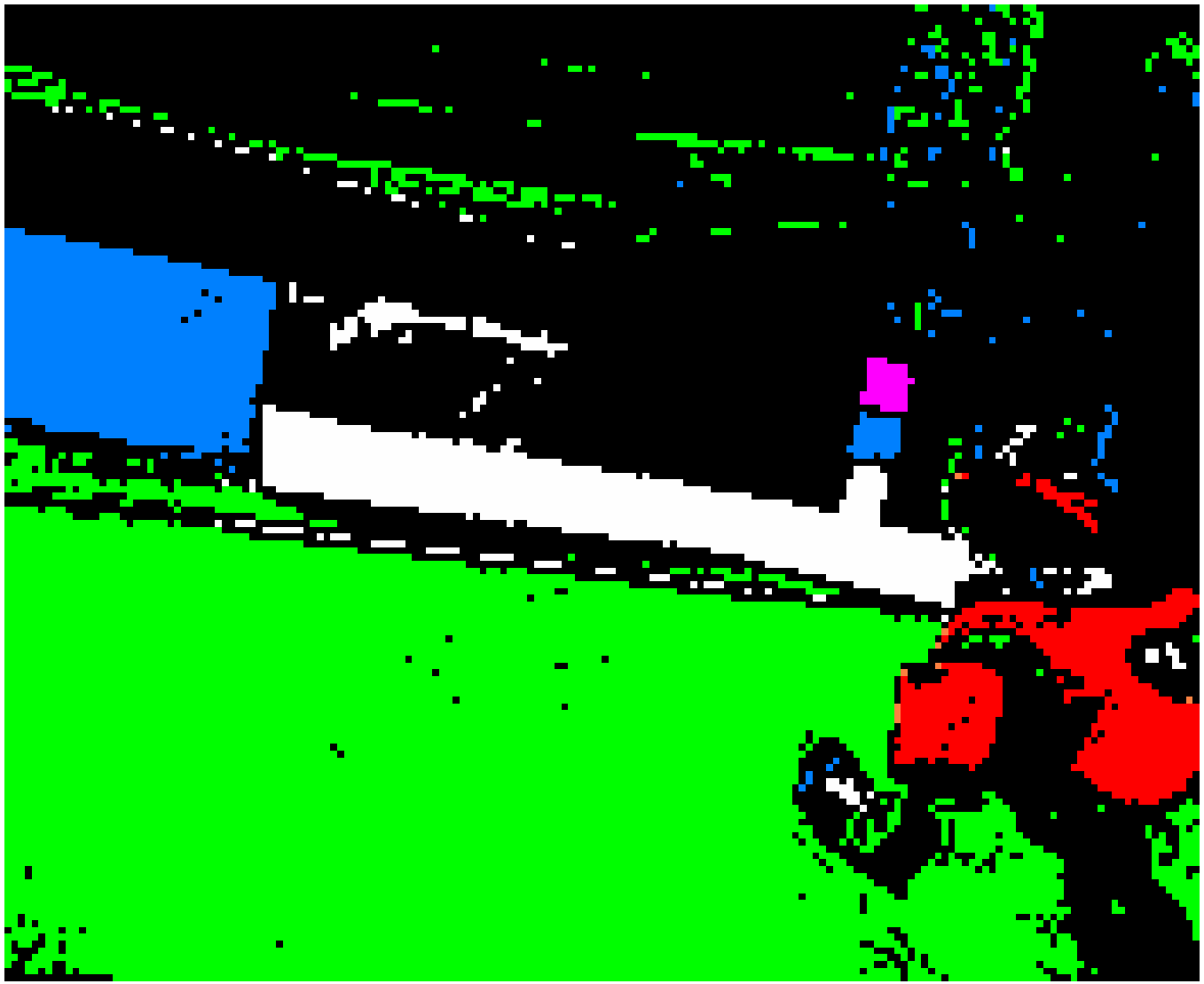
    where $\text{fix}(a) = \max(0,\min(1,a))$
- RGB → HSI
  - $i := 0.3\ r + 0.59\ g + 0.11\ b$

    or $i := (r + g + b) / 3$
  - $s := 1 - \min(r,g,b) / i$
  - $h := \text{atan2}(\sqrt{3}\ (g - b),\ 2\ r - g - b) / 2\pi$

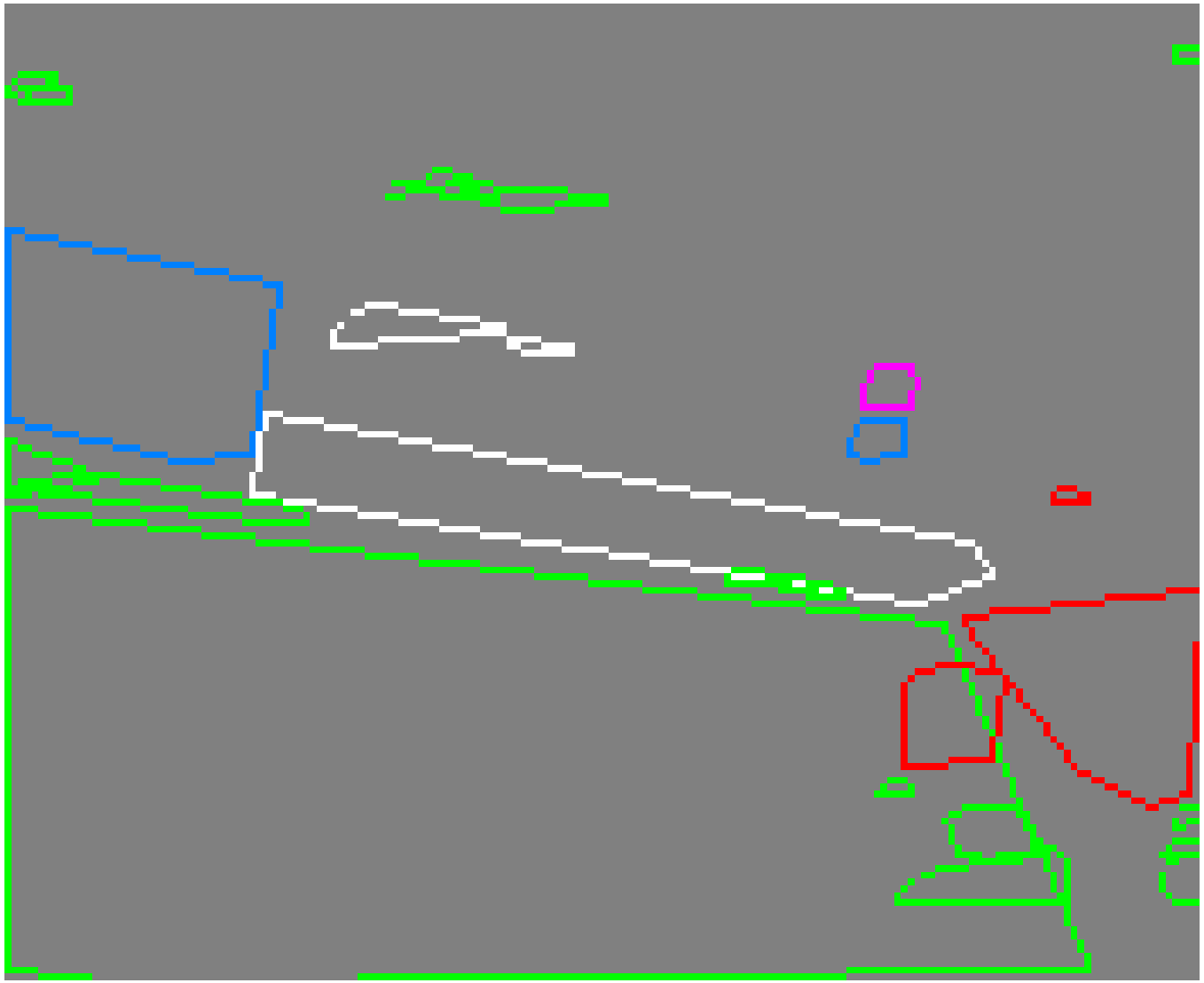


Perception – Blob-Based Vision

Perception – Blob-Based Vision

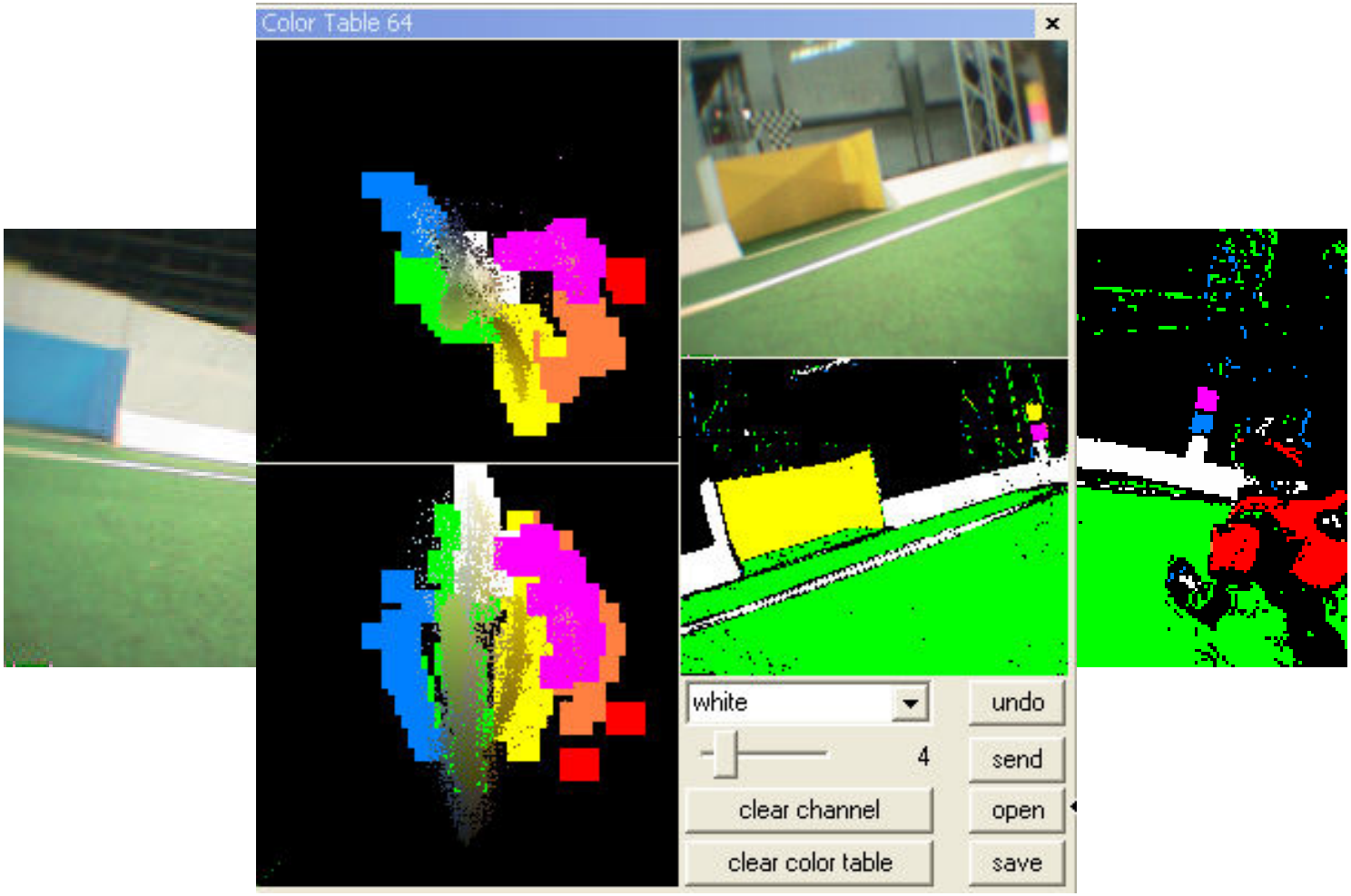


Perception – Blob-Based Vision



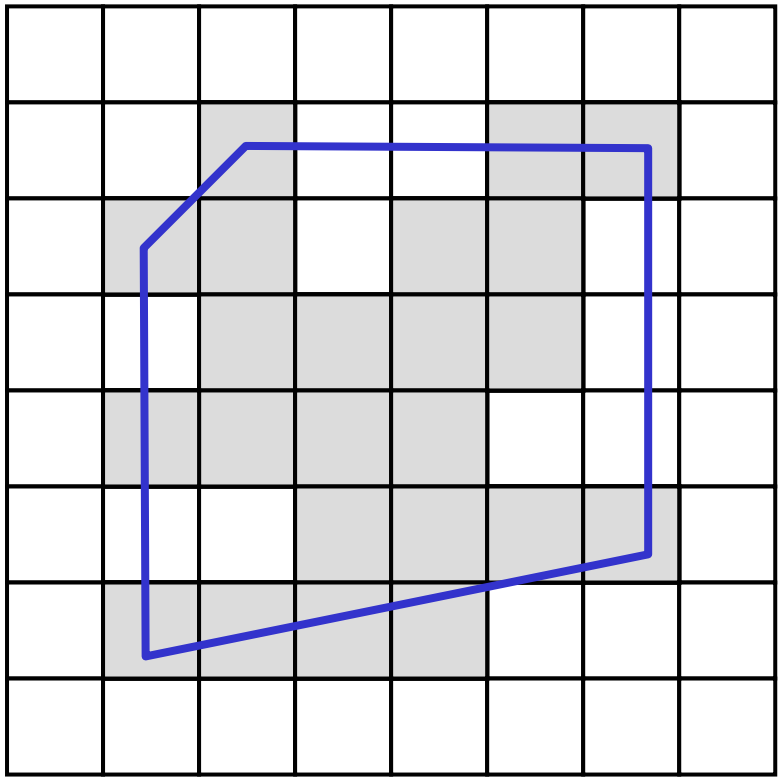
Perception – Blob-Based Vision

Perception – Color Segmentation



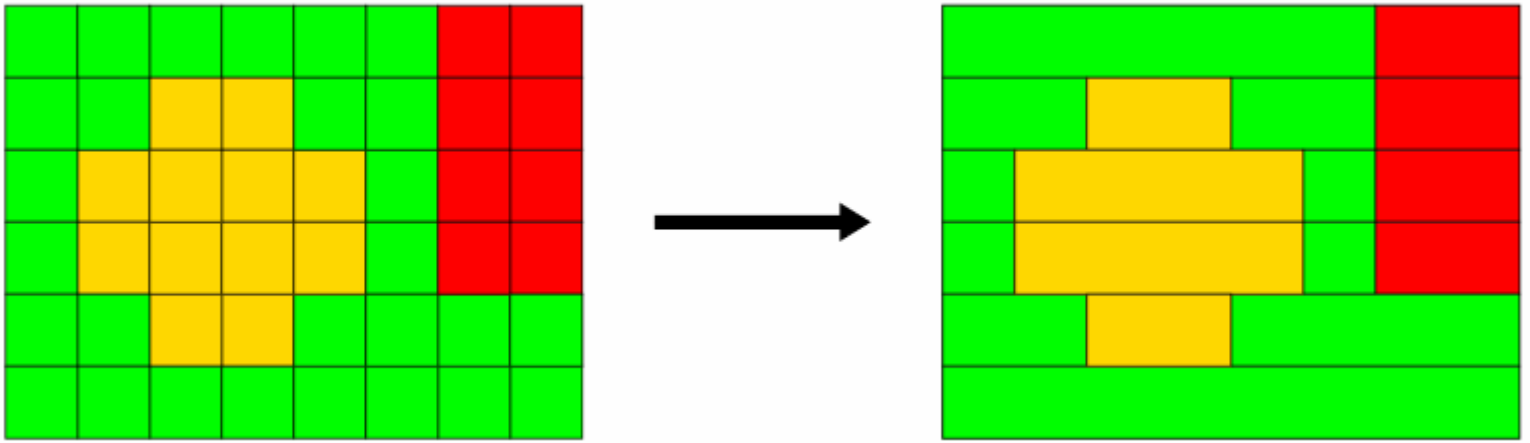
Perception – Recognition of Blobs

- Find connected areas
  - for all c(x,y)
    if c(x,y) ≠ noColor
      fill pixels of class c(x,y)
        starting at (x,y)
- fill (c,x,y)
  - init boundary
    push (x,y) on stack
    clear c(x,y)
  - while stack not empty
    pop (x‘,y‘) from stack
    update boundary
    for each neighbor (x“,y“)
     of (x‘,y‘) with c(x“,y“) = c
      push (x“,y“) on stack
      clear c(x“,y“)



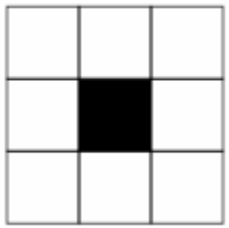
Example (4-connectedness)

Perception – RLE Compression

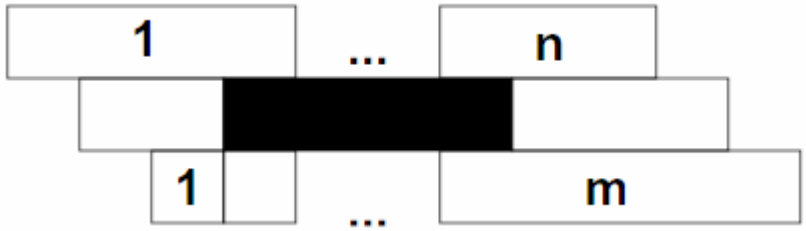


Normal image

Run-length encoded image



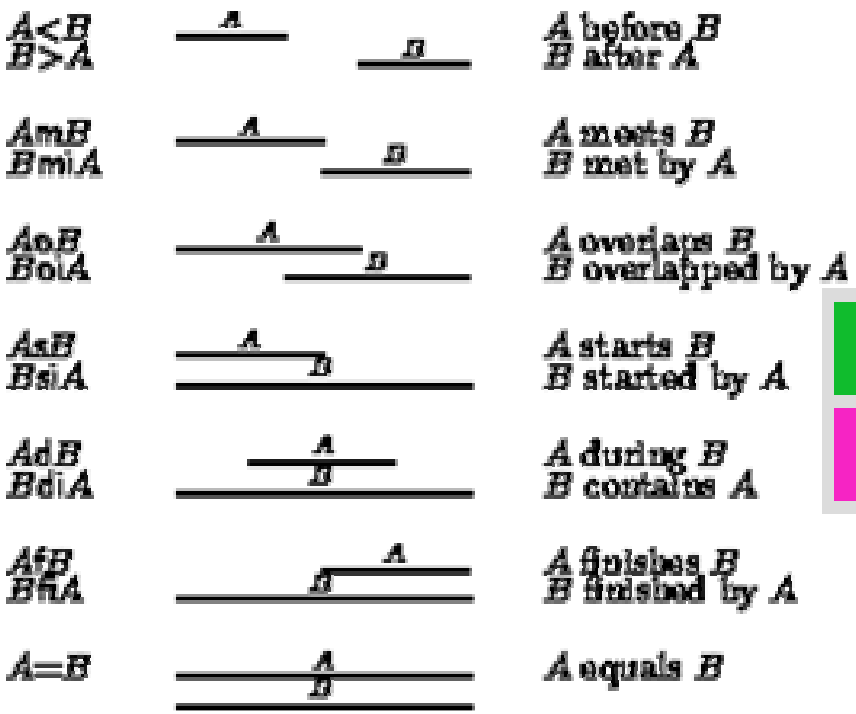
8 neighbors

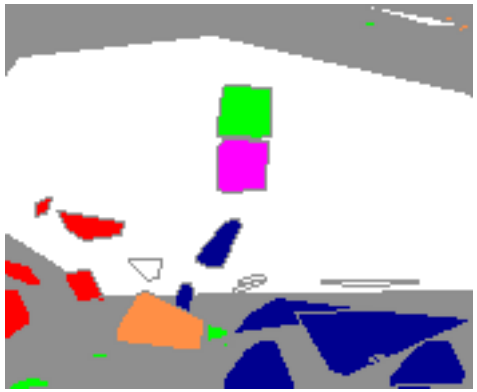


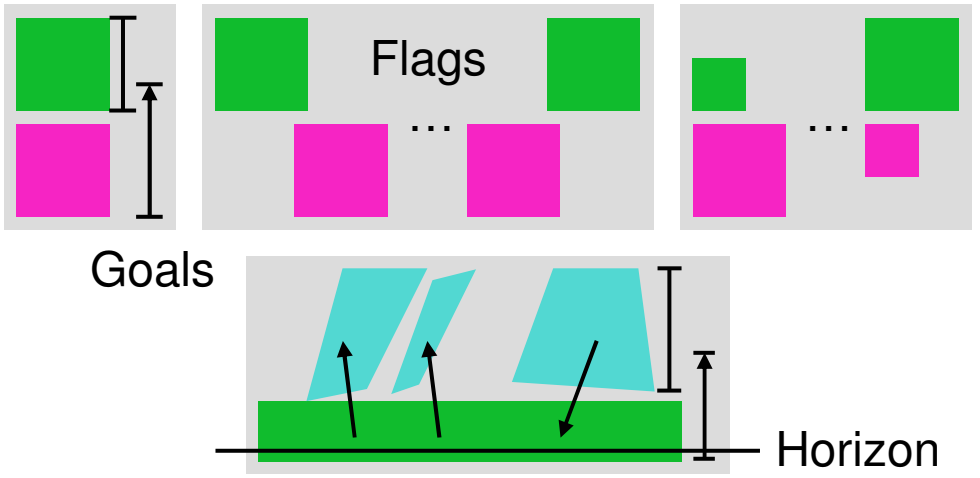
n+m neighbors

Perception – Blob Combination

- Constraints: Allen-Relations

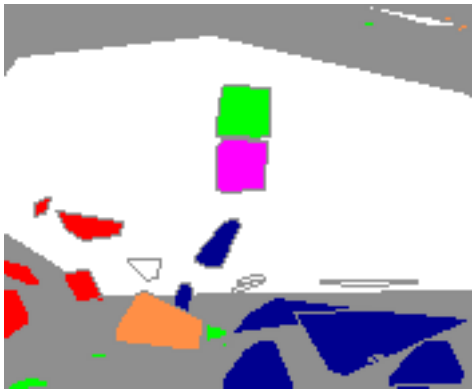


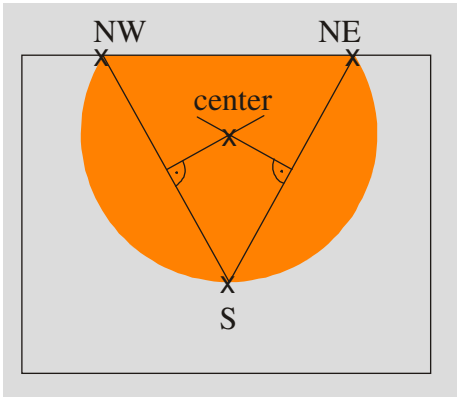


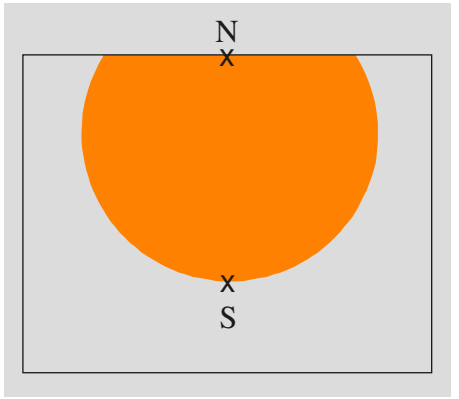


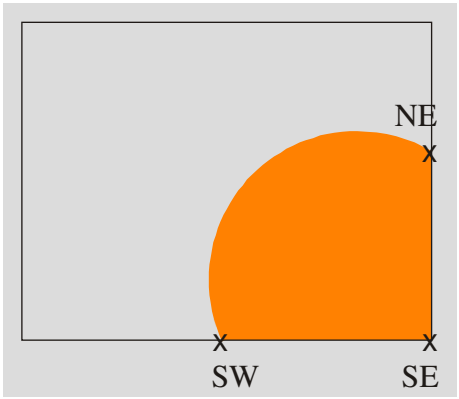
Perception – Ball and Players

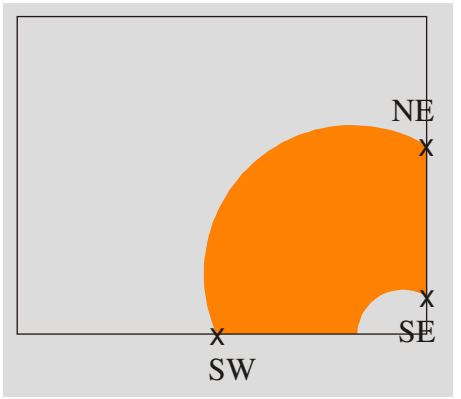
- Ball
  - Orange blob, either large or close to green
  - Selection of boundary points
  - Intersection of middle perpendiculars
- Players
  - Cluster red and blue blobs
  - Find largest blob in cluster
  - Calculate distance from height



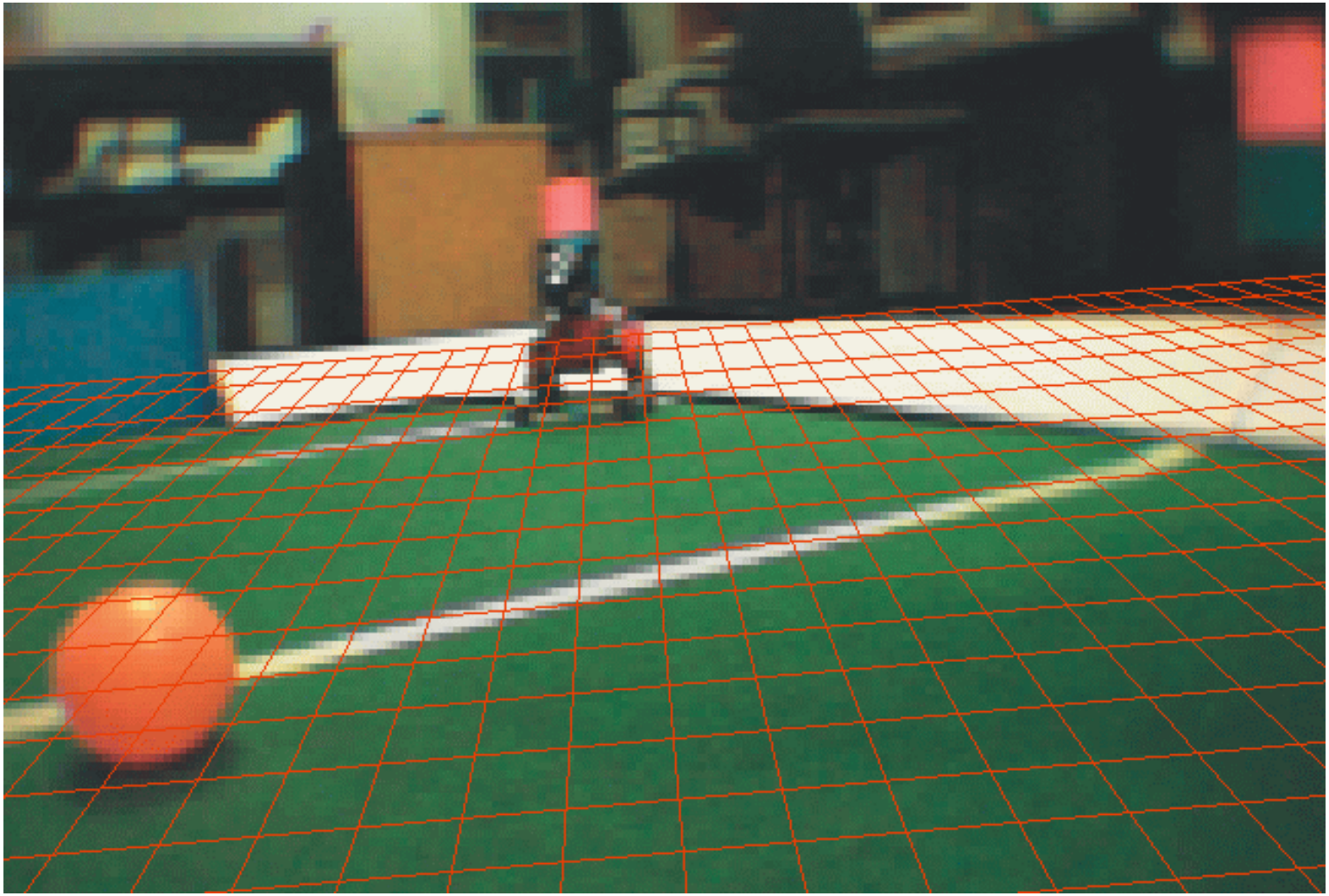




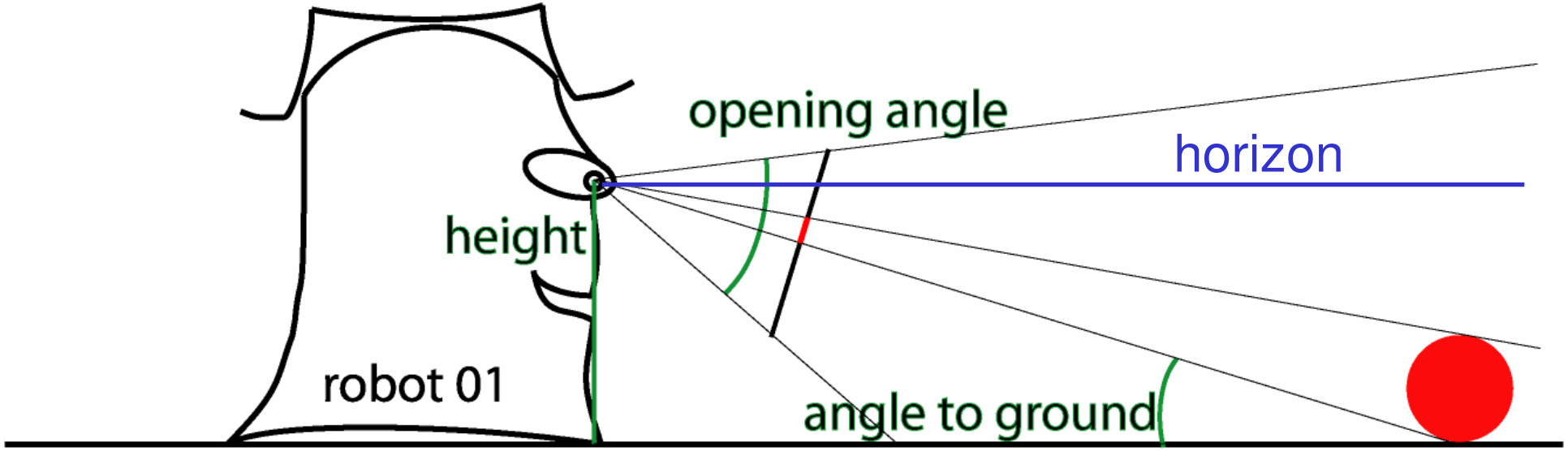




Perception – Alternative Approach



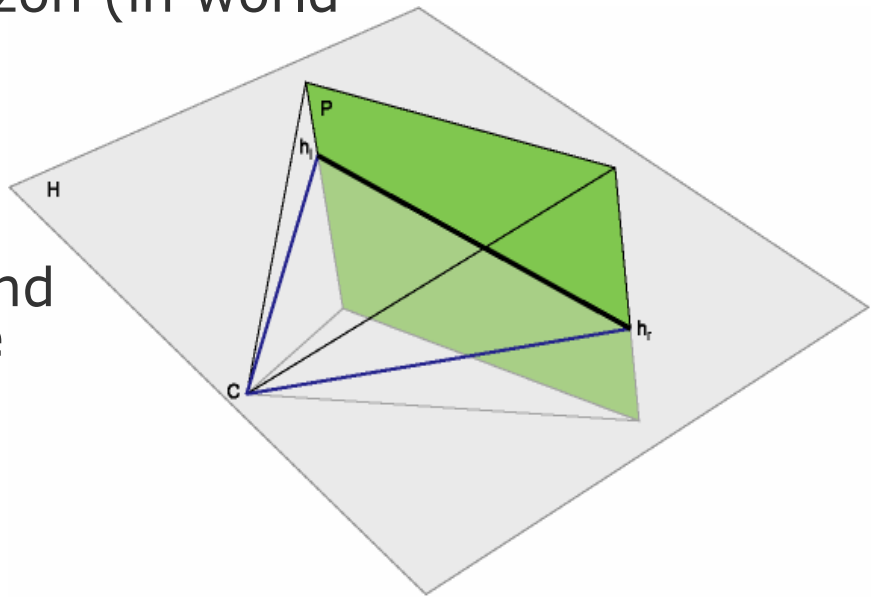
Perception – Calculation of the Horizon (1)



Perception – Calculation of the Horizon (2)

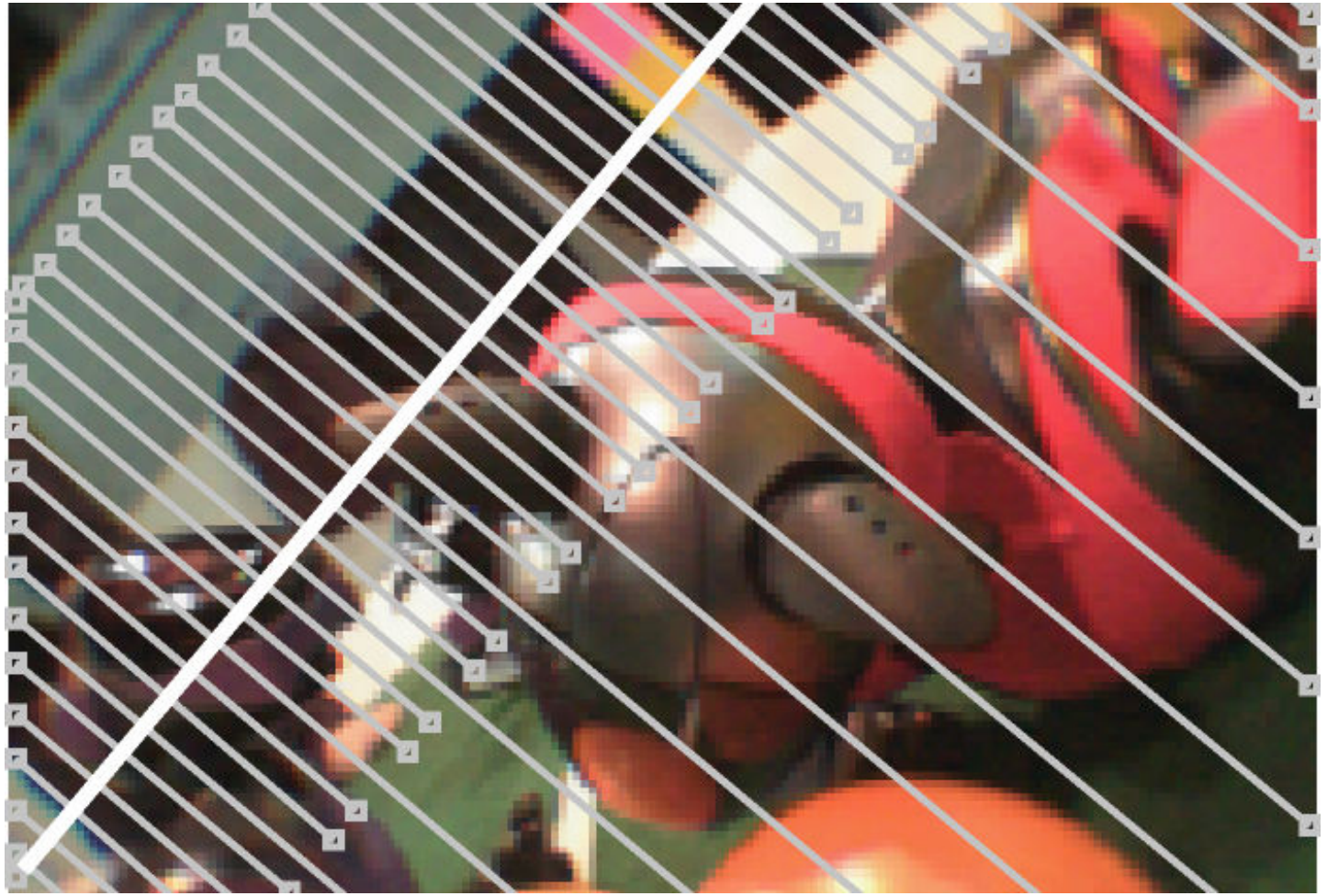
- Vectors in direction of the horizon (in world coordinates):
  $w = (w_x, w_y, 0)$
- In camera coordinates:
  $w^* = M_{camera}.rotation\ w$
- With opening angle being 2α and camera resolution s, $h_{l/r}$ can be calculated as:

$$h_l = \begin{pmatrix} \frac{s}{\tan\alpha} \\ s \\ z_l \end{pmatrix}, h_r = \begin{pmatrix} \frac{s}{\tan\alpha} \\ -s \\ z_r \end{pmatrix}$$



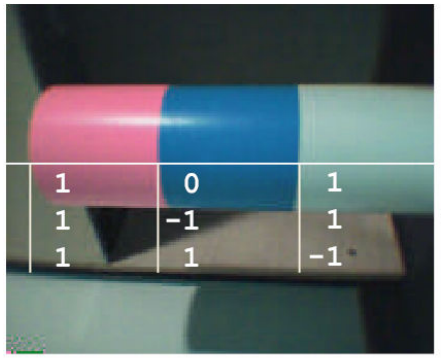
- If h_l and h_r are on the horizon, they satisfy
  $h_{l/r} = w^* = M_{camera}.rotation\ w$
- This can easily be solved for $z_{l/r}$, resulting in the intersection points of the horizon with the image borders

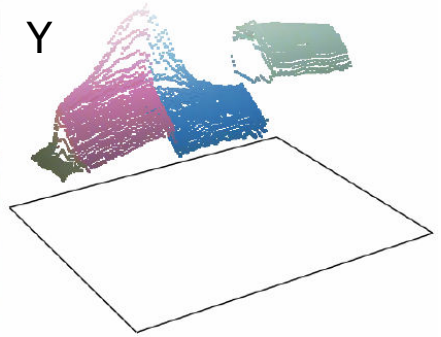
Perception – Grid Perpendicular to Horizon

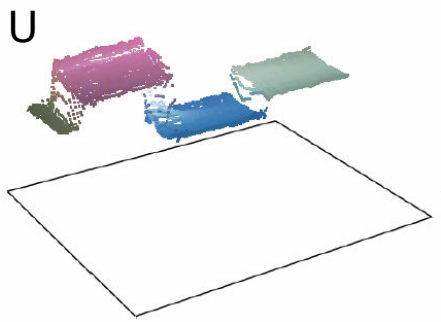


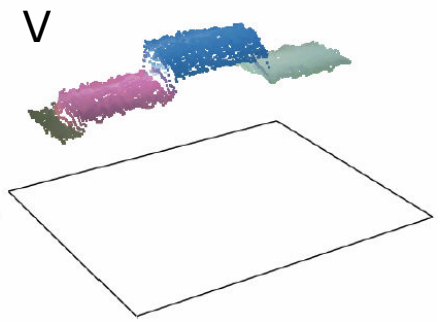
Perception – Object Recognition

- Edge detection along scan lines is used for object recognition
- Color class (not all colors classifyable)
- Patterns in contrast in the YUV channels are used to resolve color ambiguities
- Anchor points for specialists

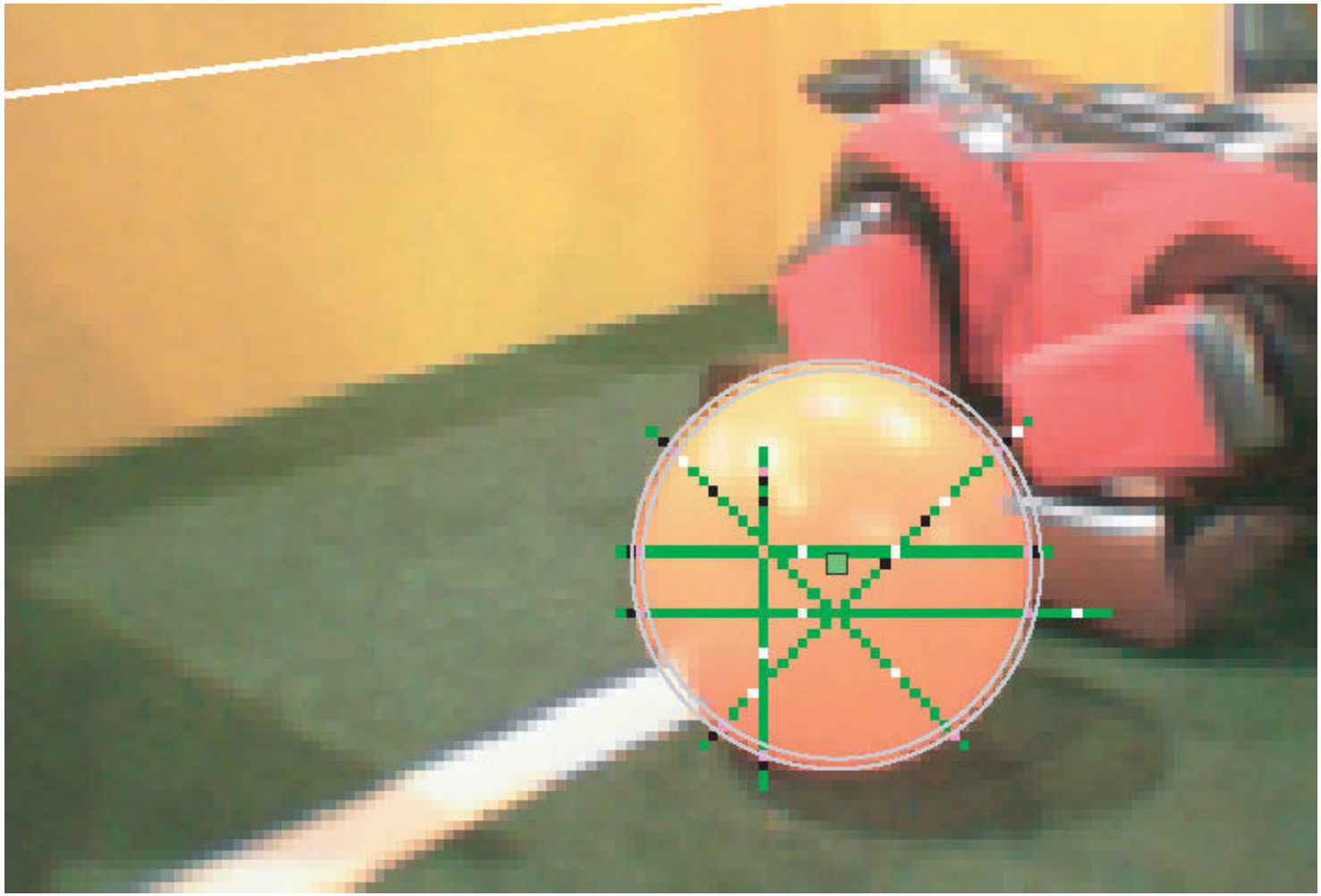




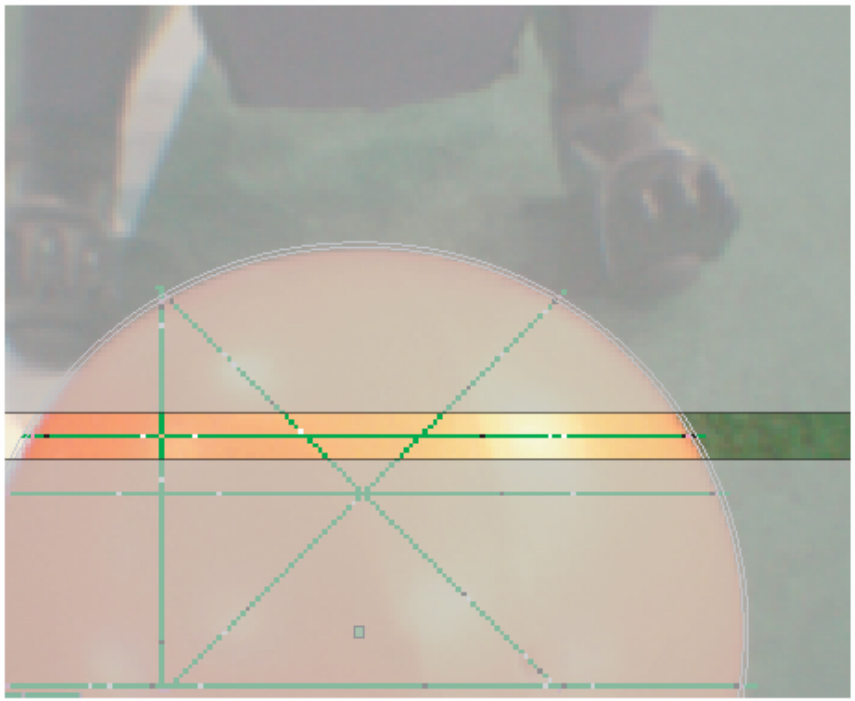


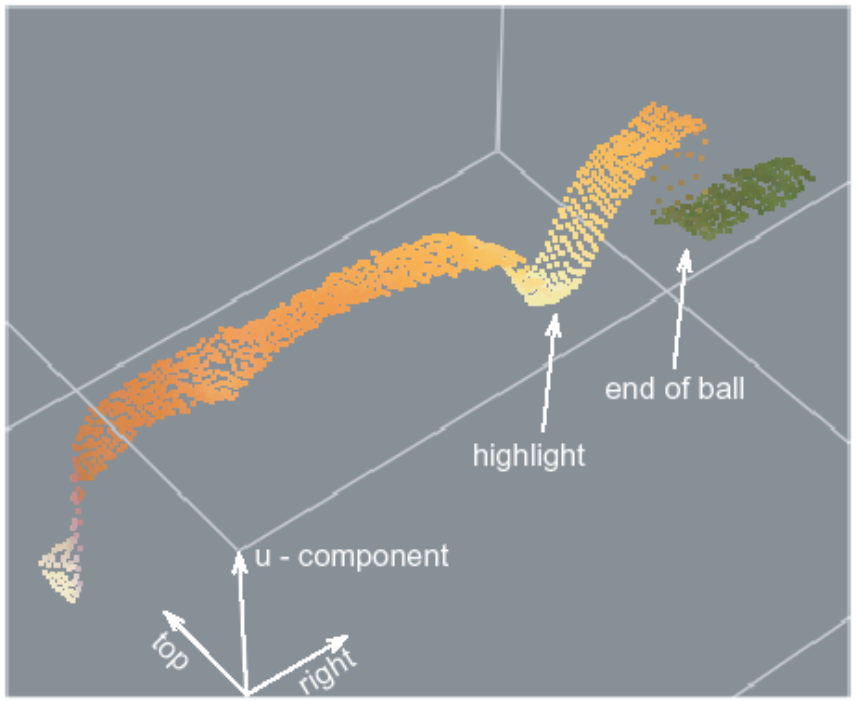


Perception – Specialists



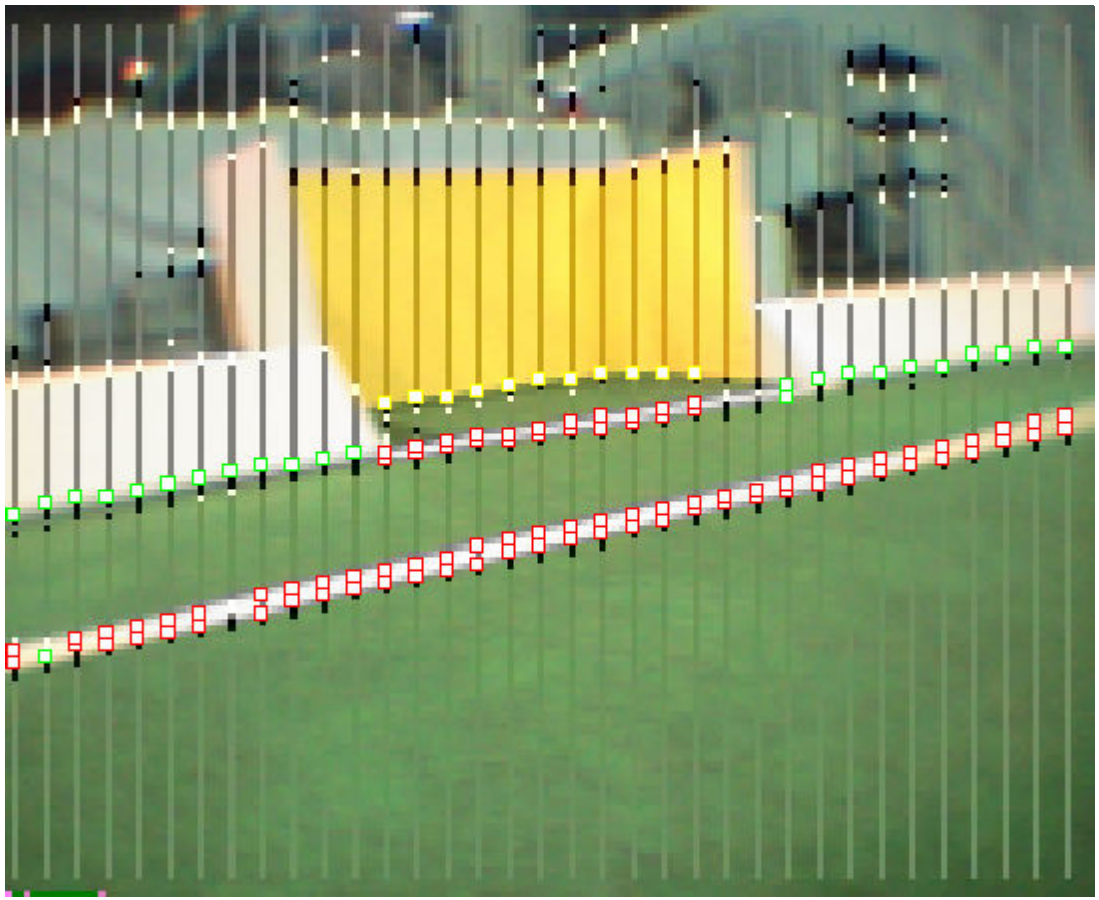
Perception – Using Gradients



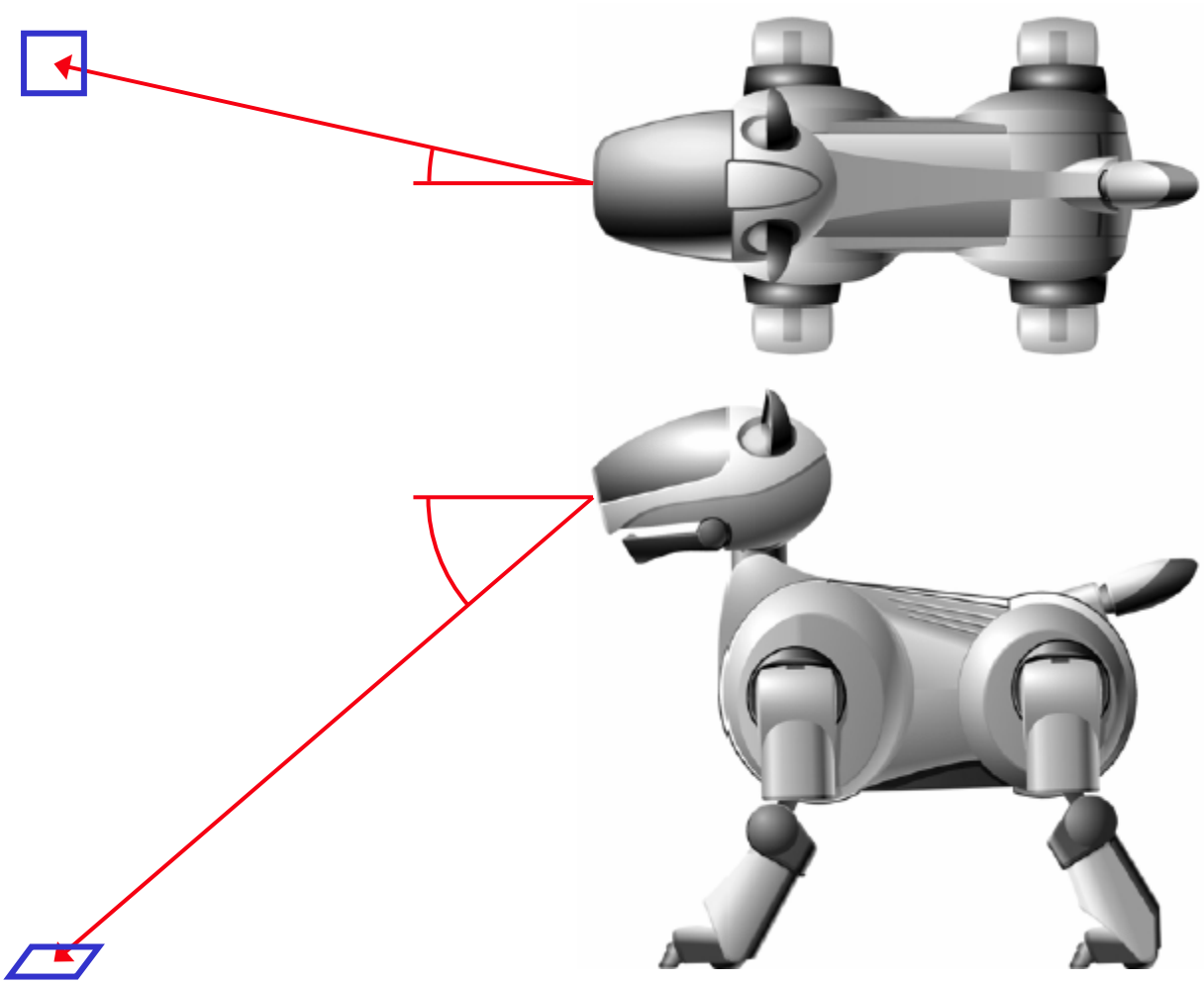


Perception – Detecting Edges

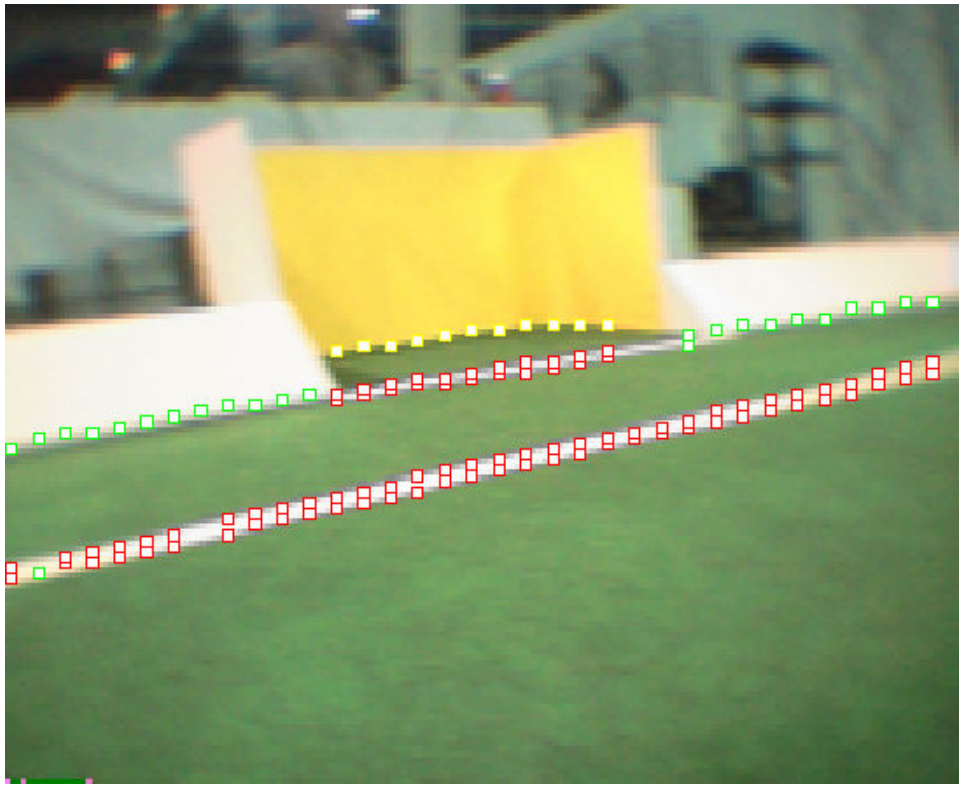
- Between field and
  - Border
  - Field lines
  - Goals
    - *yellow*
    - *skyblue*

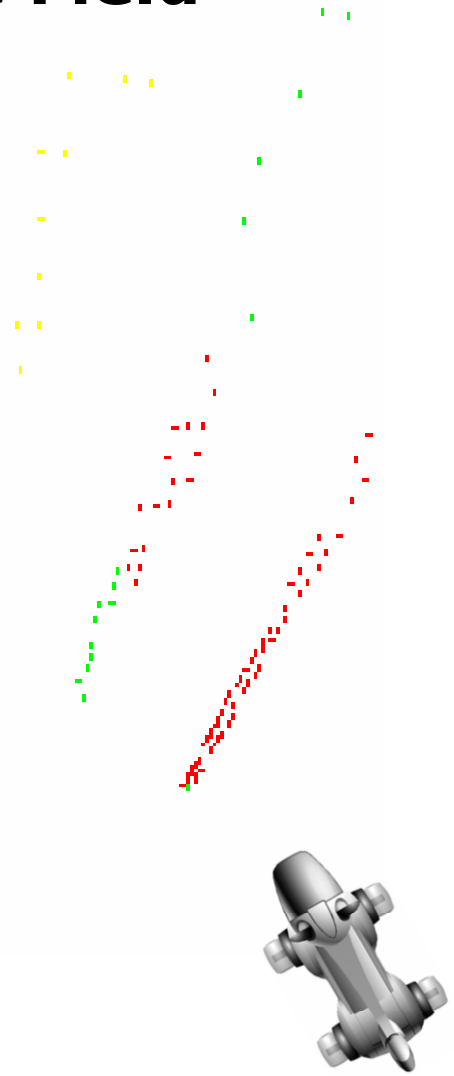


Perception – Distances

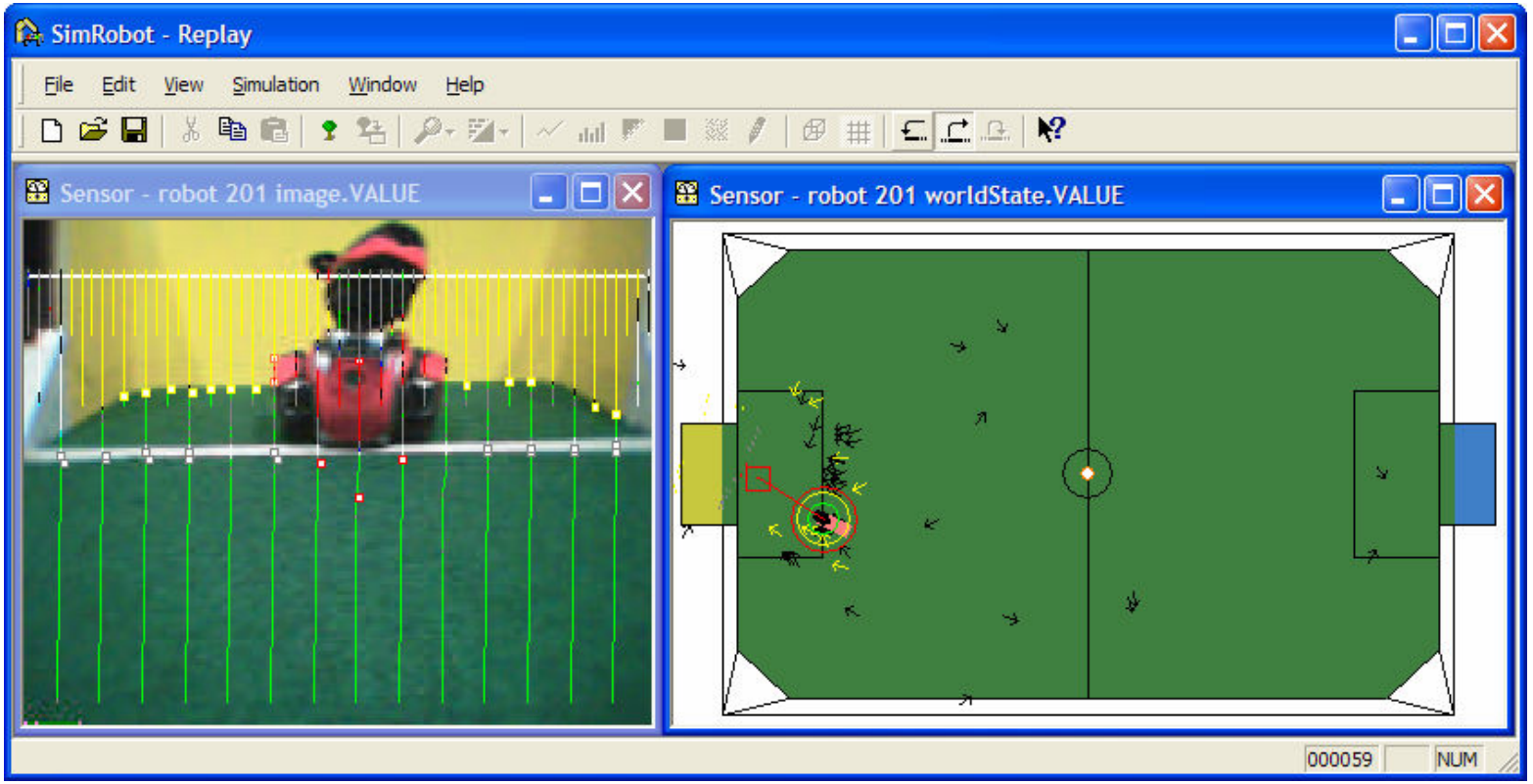


Perception – Projection on the Field

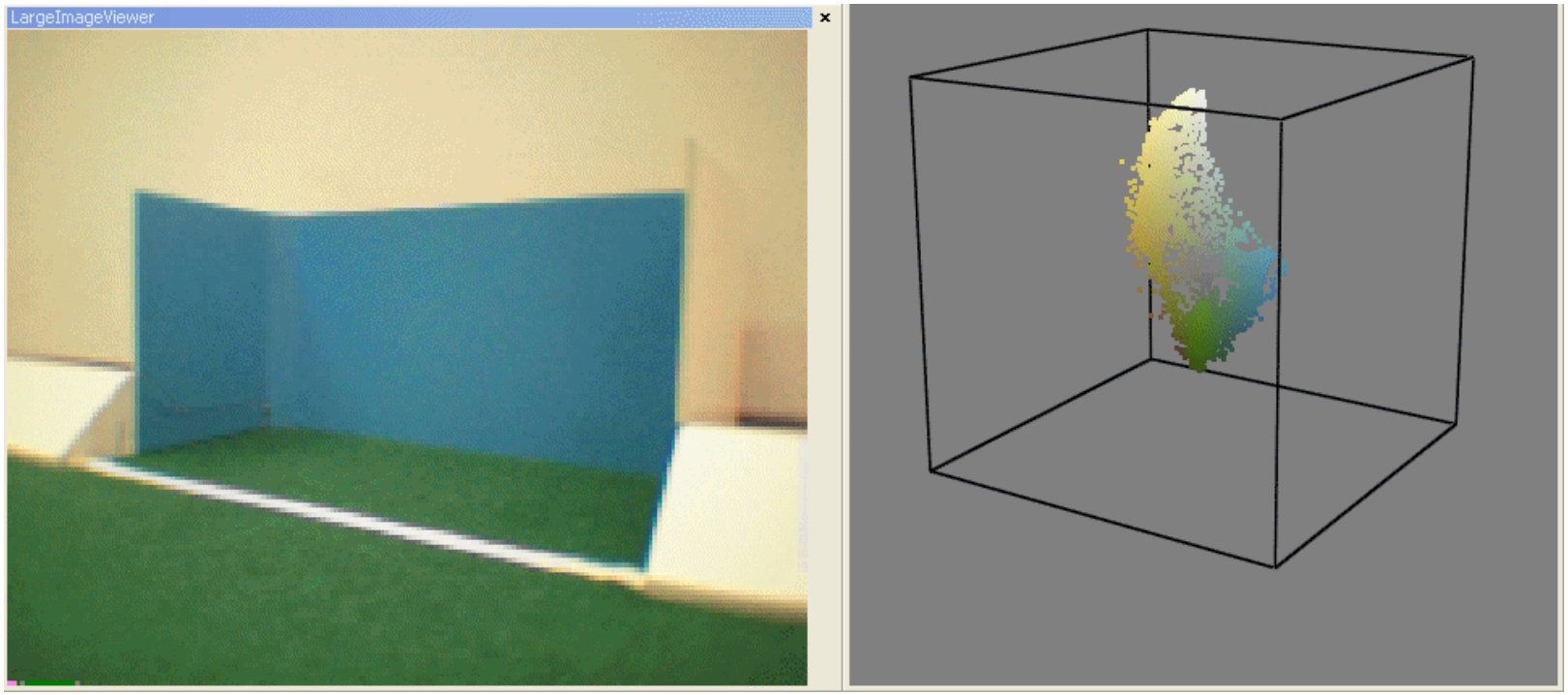




Perception – Example

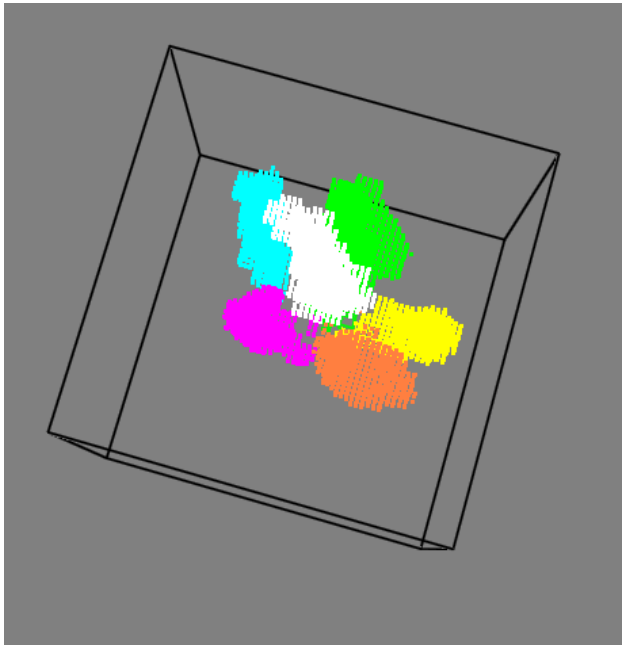
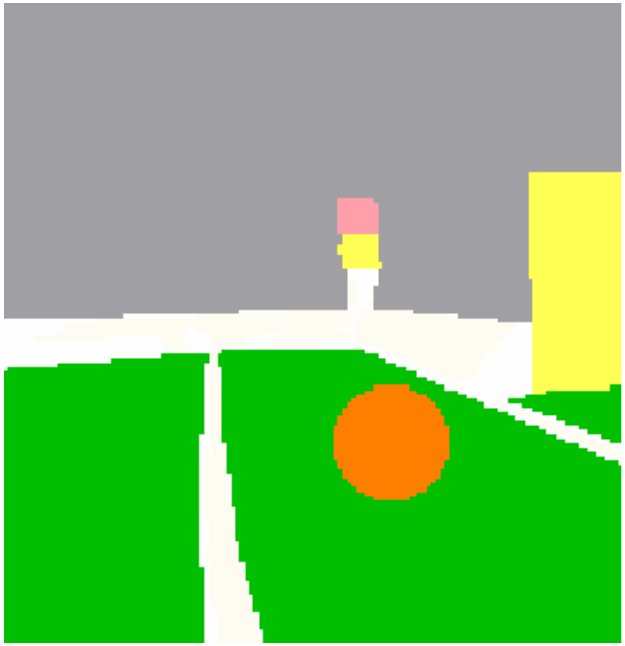


Perception – Different Lighting Conditions

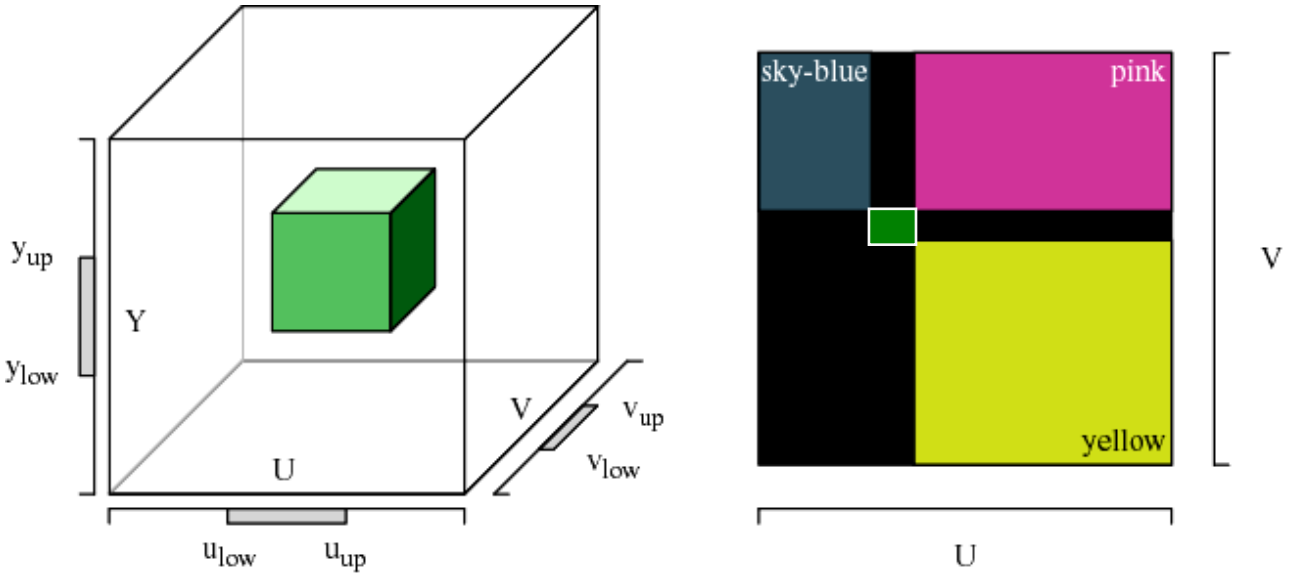


Color Labeled Environment
→ Color Classification

Perception – Color Segmentation

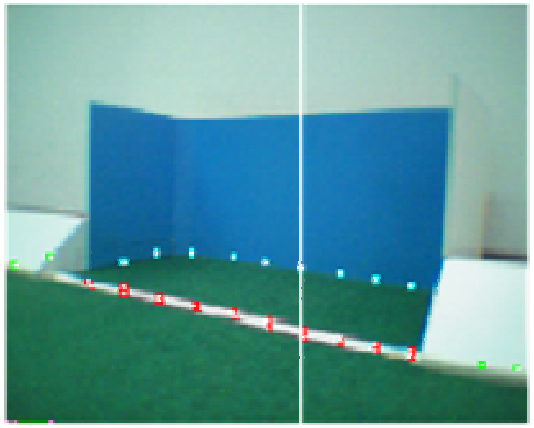




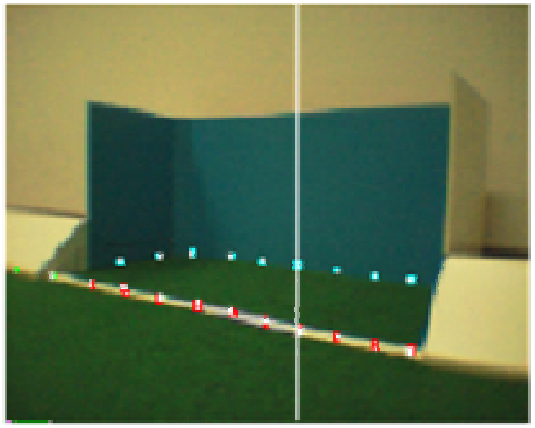
Perception – Classification Relative to a Reference Color

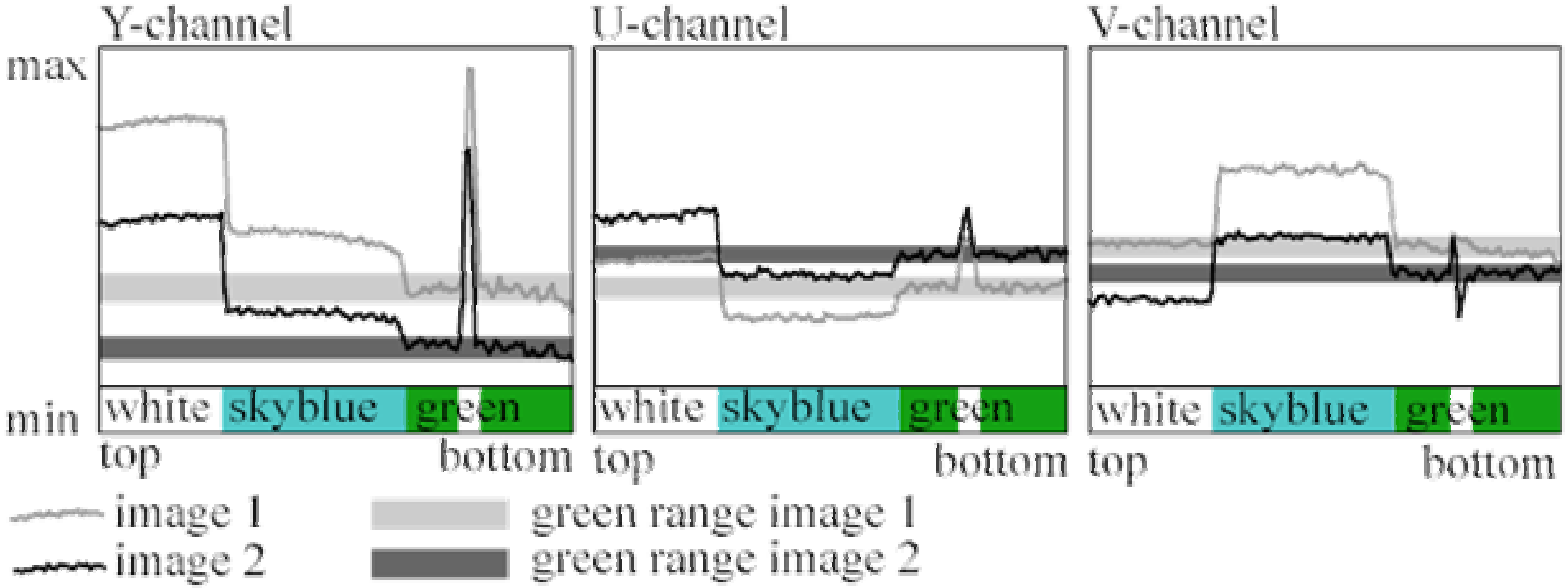


Colors are classified relative to green in the UV plane

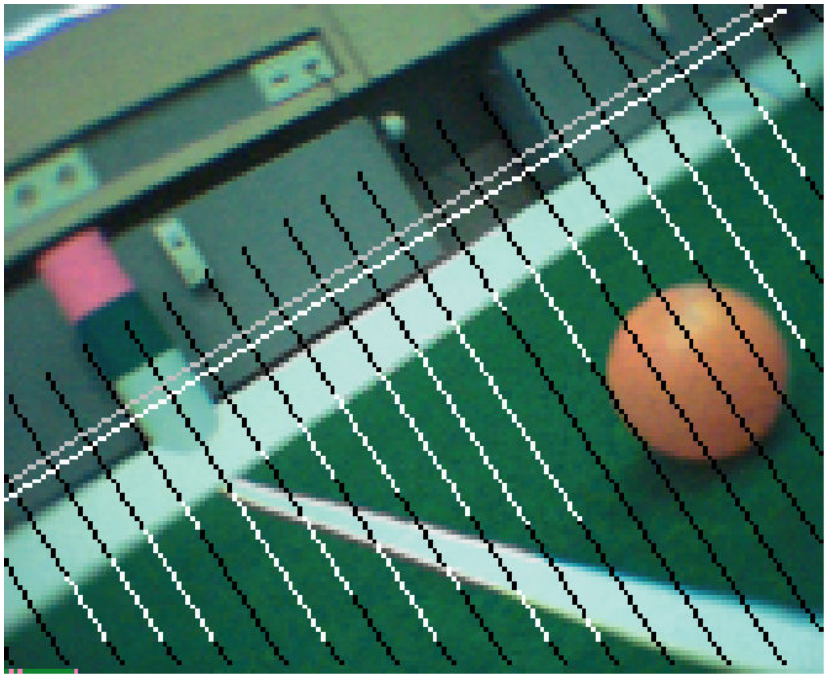
Perception – Reference Color Green





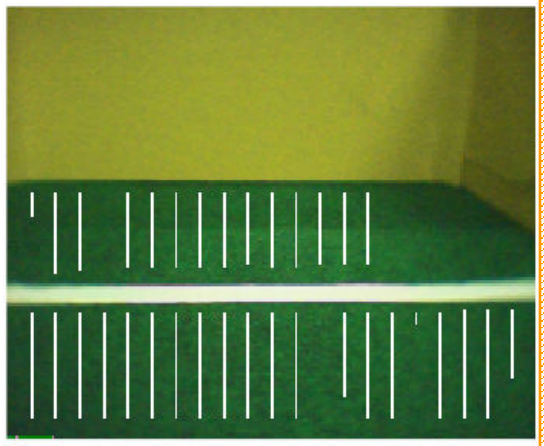


Perception – Calibration Using Knowledge about the Environment

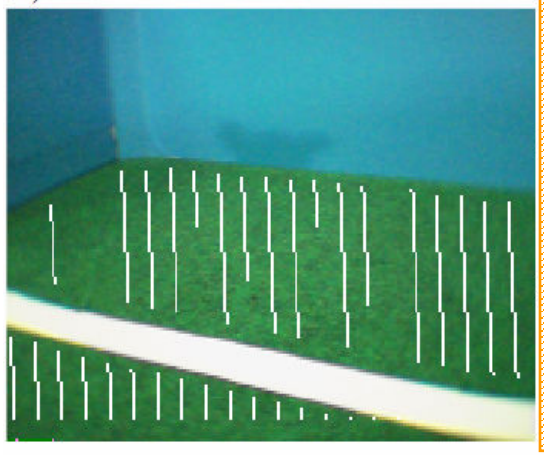


Green pixels can be identified by characteristic edges along scan lines in the YUV channels

Perception – Color Histogram



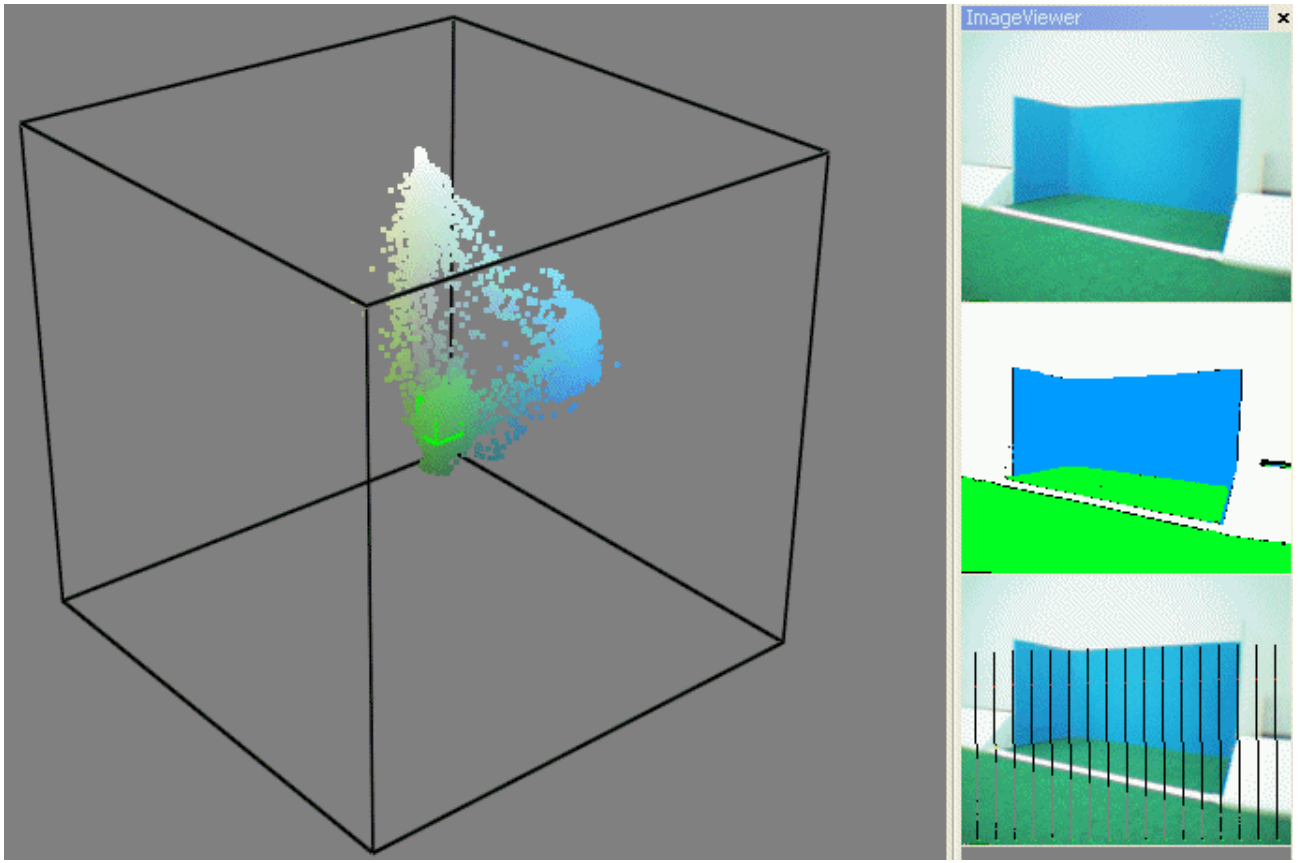
b)



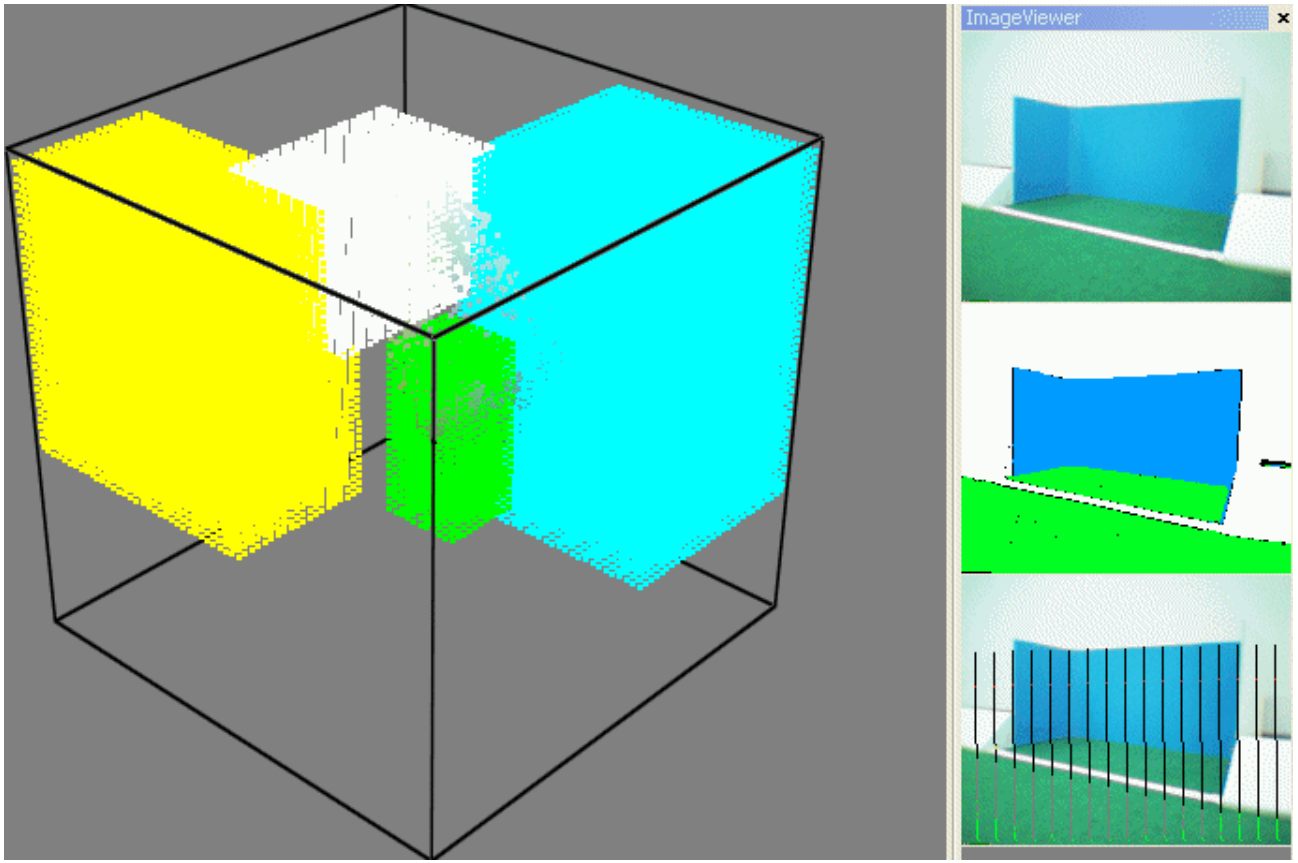
Histogram over pixels used for calibration

Histogram of all image pixels

Perception – Auto Adaptation



Auto Adaptation



Localization in the Sony Four-Legged Robot League

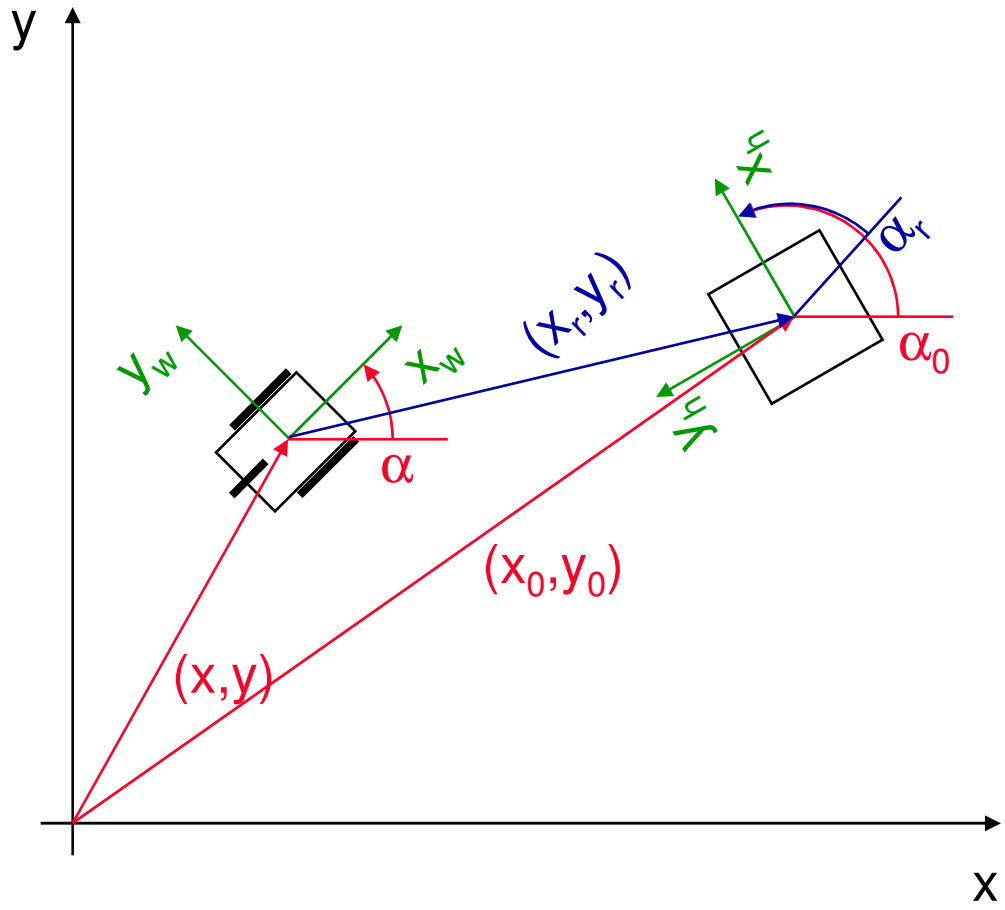
- Advantages
  - Automatic positioning
  - Sharing perceptions
  - Full support of referee commands
- Challenges
  - Limited computing power
  - Vision-based
  - Directed vision
  - Variable camera position

Localization in the Sony Four-Legged Robot League

- Advantages
  - Automatic positioning
  - Sharing perceptions
  - Full support of referee commands
- Challenges
  - Limited computing power
  - Vision-based
  - Directed vision
  - Variable camera position

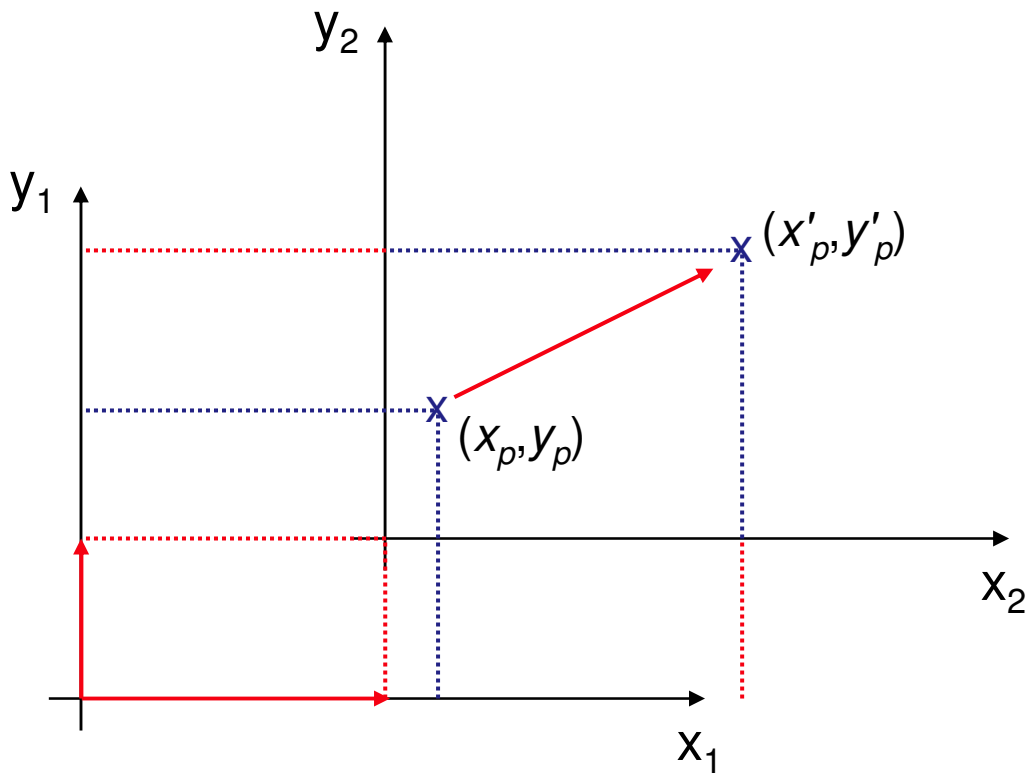
Localization – Operations on Poses

- Operations



Localization – Operations on Poses

- Operations
  - Translation

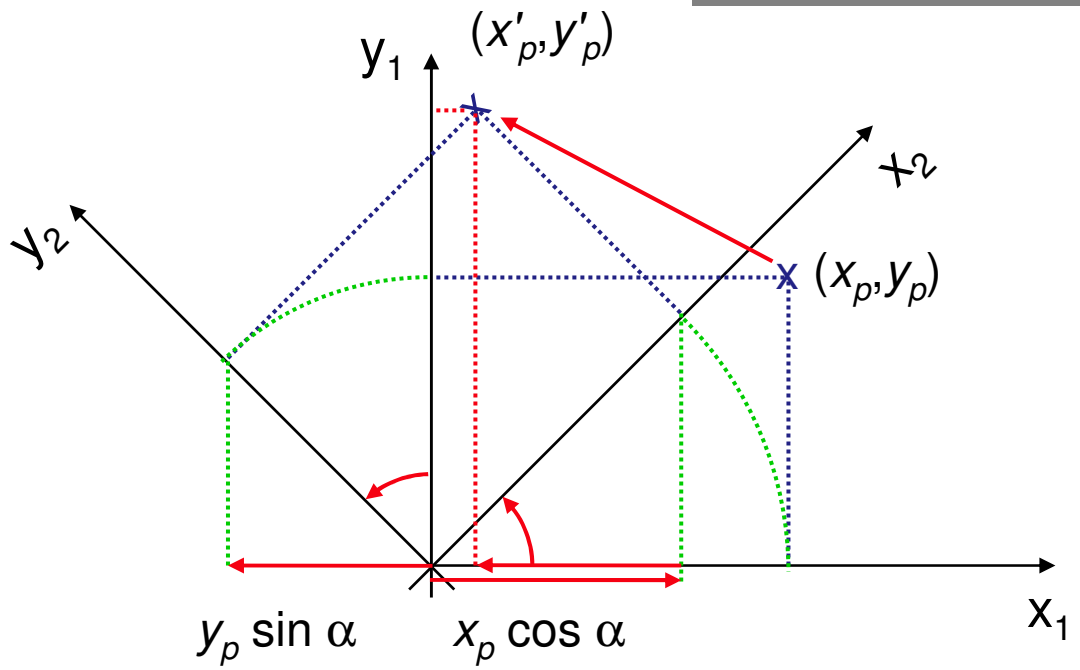


Localization – Operations on Poses

- Operations
  - Translation
  - Rotation

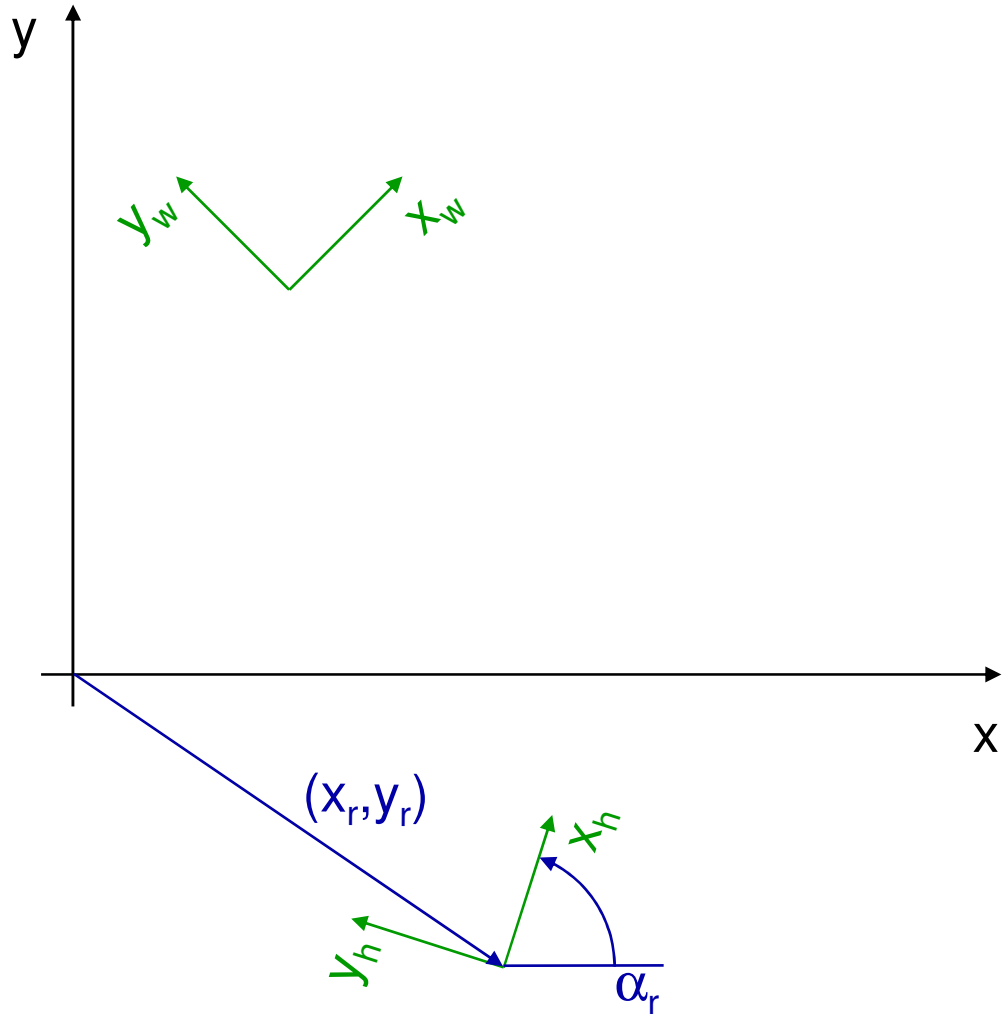
$$x'_p = x_p \cos \alpha - y_p \sin \alpha$$

$$y'_p = x_p \sin \alpha + y_p \cos \alpha$$



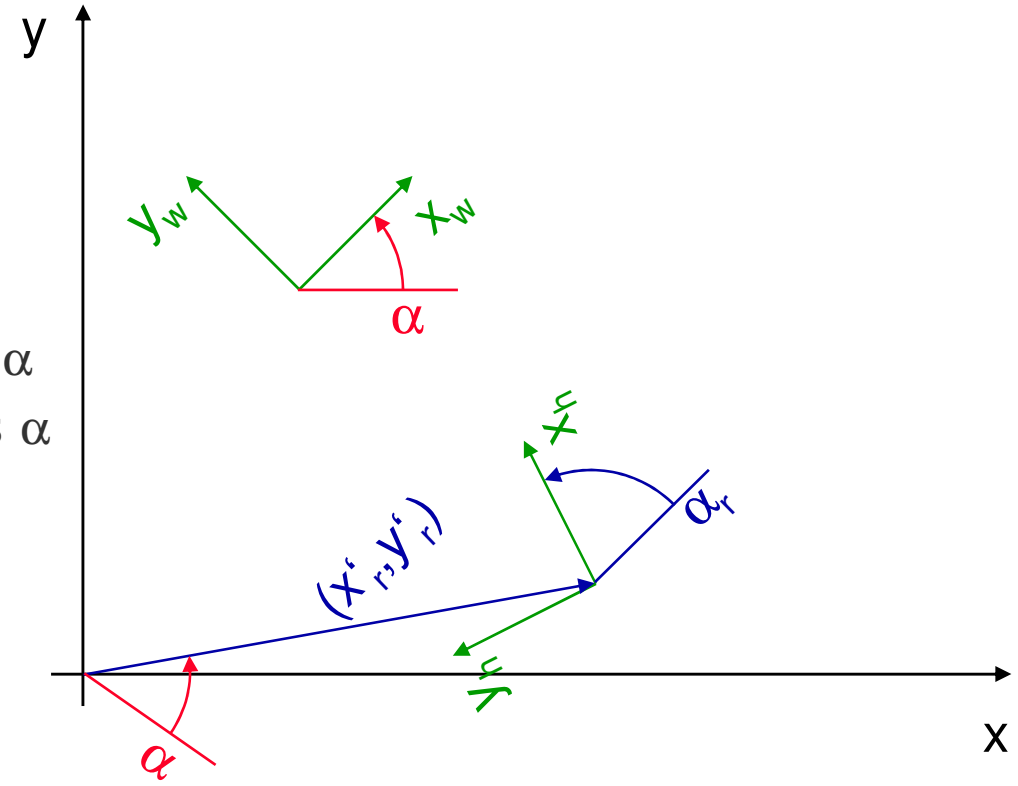
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition



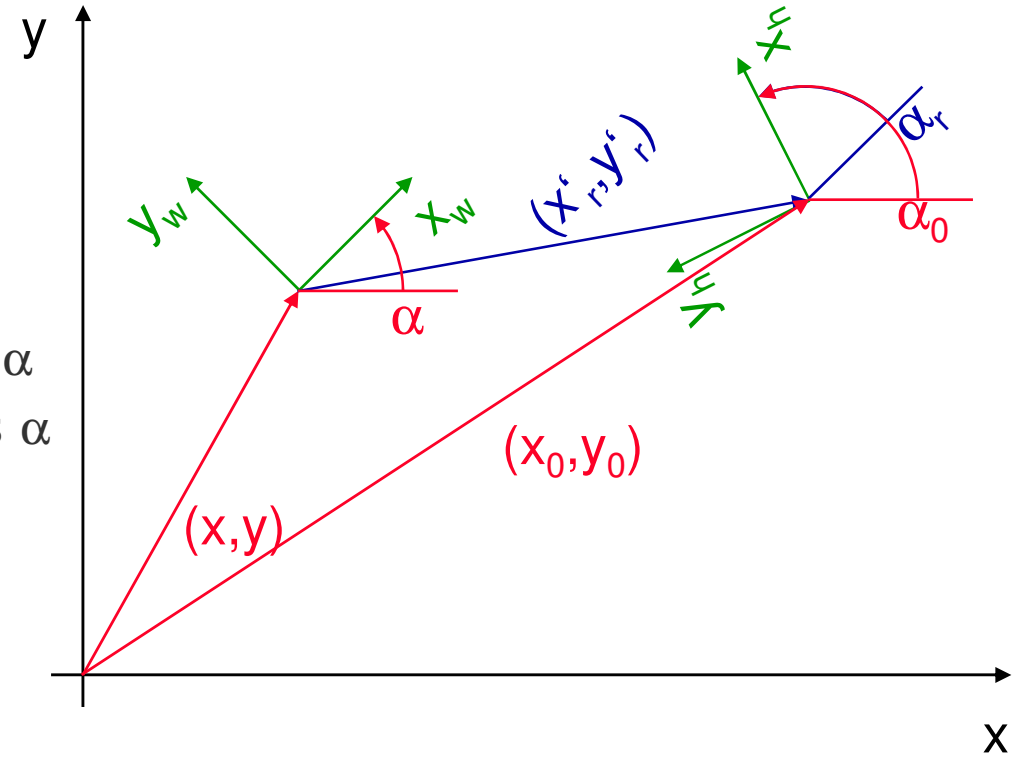
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition
  - $x'_r = x_r \cos \alpha - y_r \sin \alpha$
  - $y'_r = x_r \sin \alpha + y_r \cos \alpha$



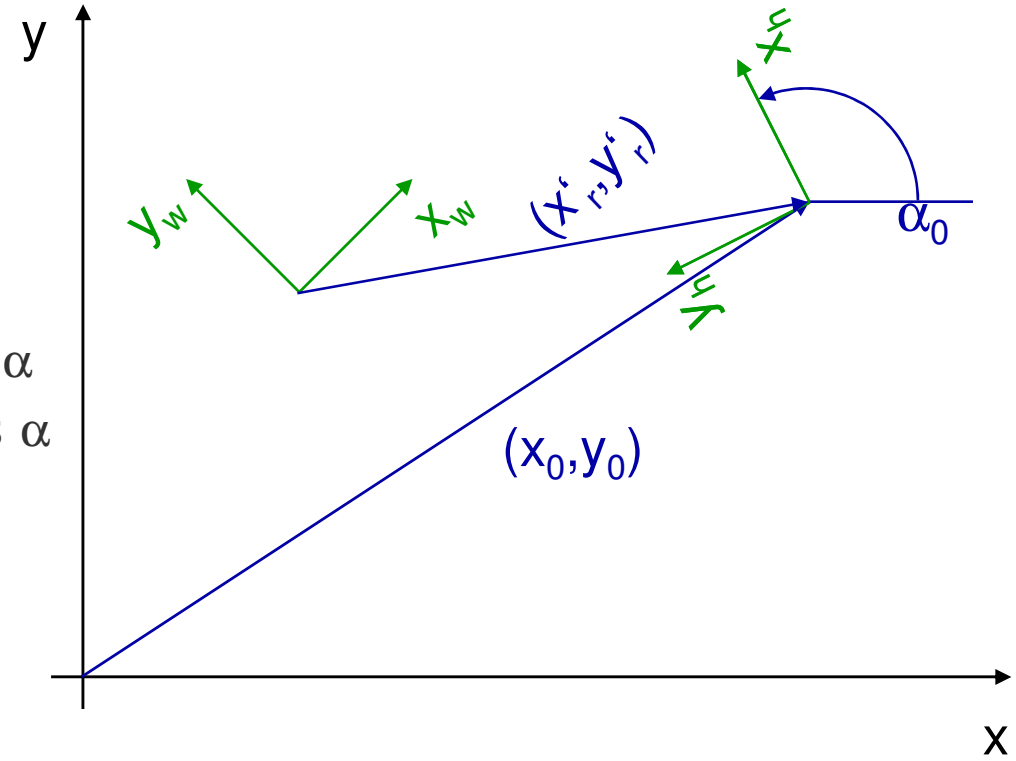
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition
  - $x`_r = x_r \cos \alpha - y_r \sin \alpha$
  - $y`_r = x_r \sin \alpha + y_r \cos \alpha$
  - $x_0 = x + x`_r$
  - $y_0 = y + y`_r$
  - $\alpha_0 = \alpha + \alpha_r$



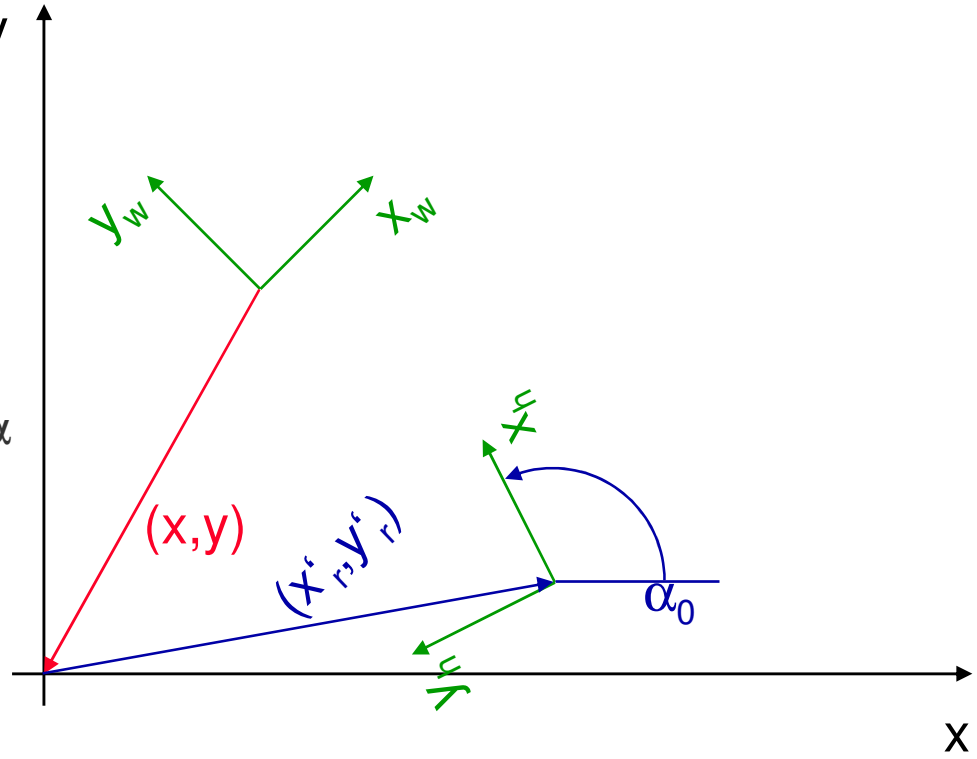
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition
  - $x`_r = x_r \cos \alpha - y_r \sin \alpha$
  - $y`_r = x_r \sin \alpha + y_r \cos \alpha$
  - $x_0 = x + x`_r$
  - $y_0 = y + y`_r$
  - $\alpha_0 = \alpha + \alpha_r$
- Subtraction



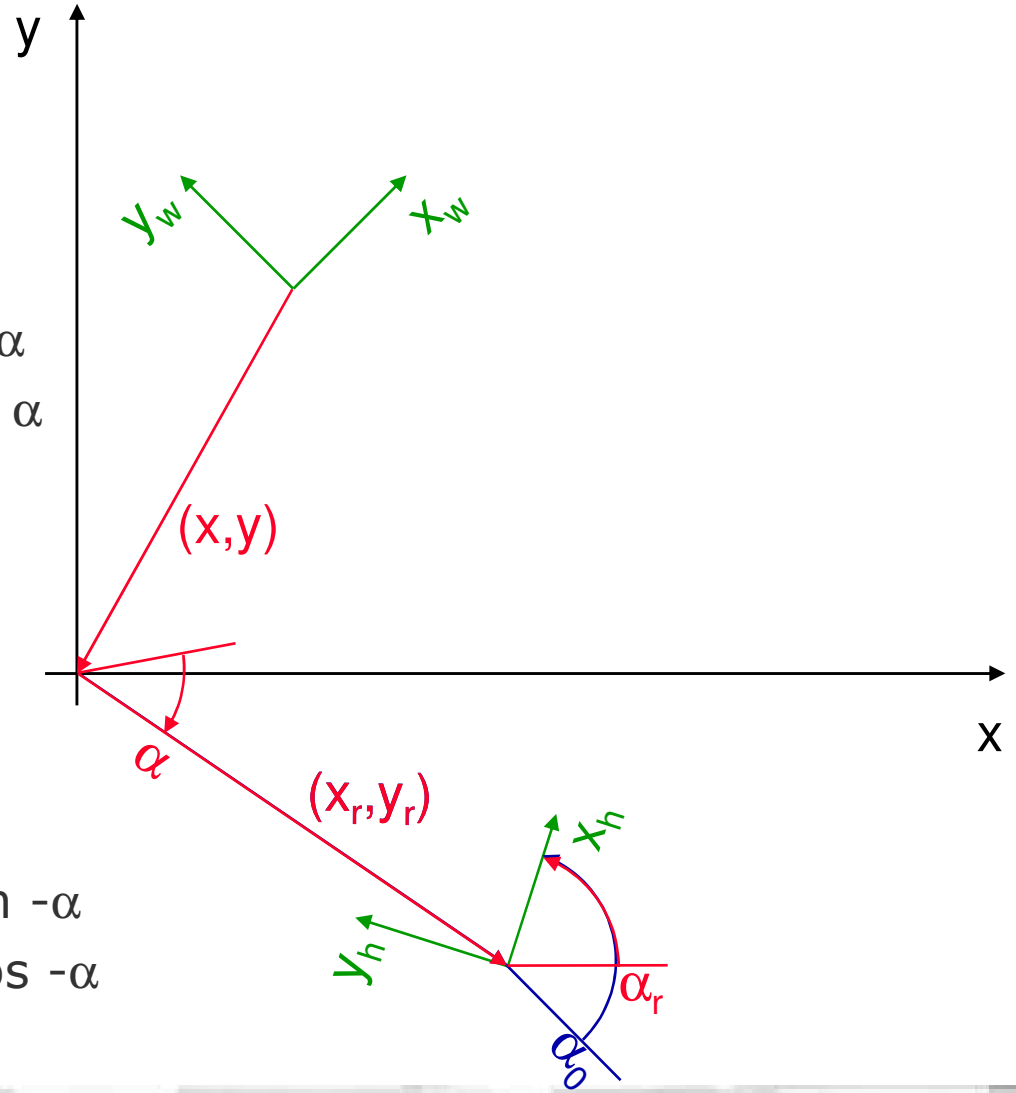
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition
  - $x'_r = x_r \cos \alpha - y_r \sin \alpha$
  - $y'_r = x_r \sin \alpha + y_r \cos \alpha$
  - $x_0 = x + x'_r$
  - $y_0 = y + y'_r$
  - $\alpha_0 = \alpha + \alpha_r$
- Subtraction
  - $x'_r = x_0 - x$
  - $y'_r = y_0 - y$

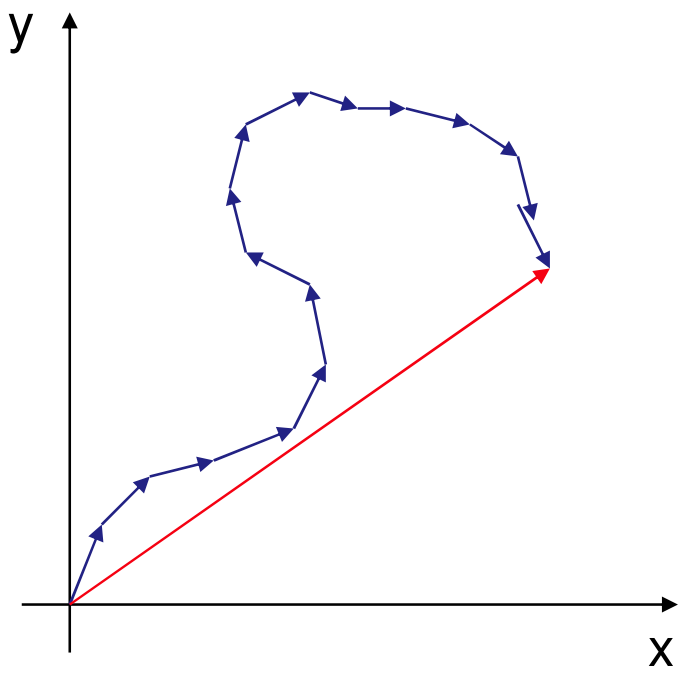


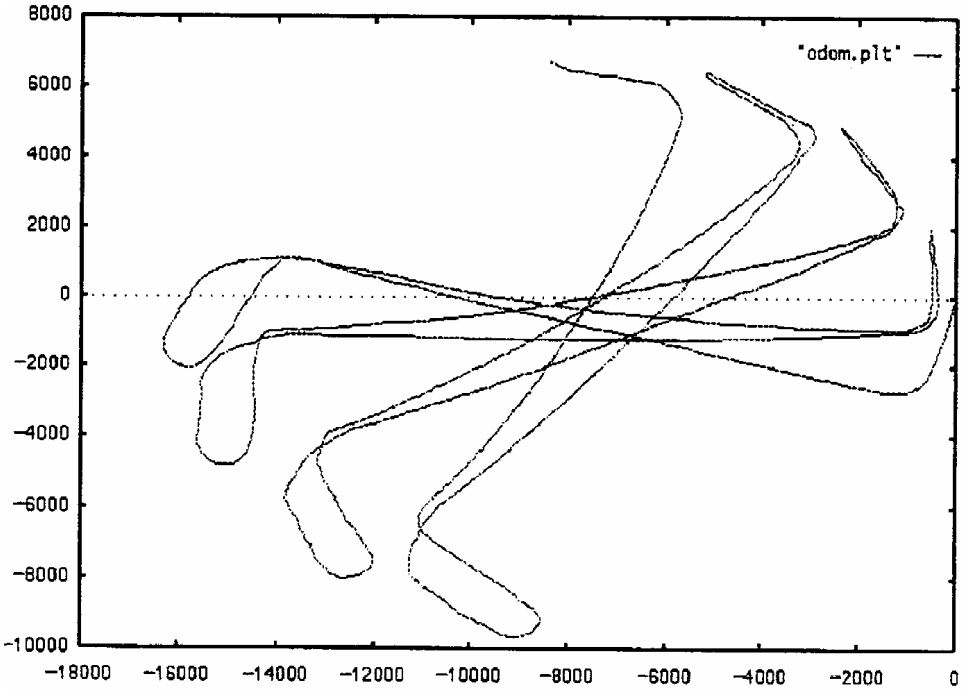
Localization – Operations on Poses

- Operations
  - Translation
  - Rotation
- Addition
  - $x`_r = x_r \cos \alpha - y_r \sin \alpha$
  - $y`_r = x_r \sin \alpha + y_r \cos \alpha$
  - $x_0 = x + x`_r$
  - $y_0 = y + y`_r$
  - $\alpha_0 = \alpha + \alpha_r$
- Subtraction
  - $x`_r = x_0 - x$
  - $y`_r = y_0 - y$
  - $x_r = x`_r \cos -\alpha - y`_r \sin -\alpha$
  - $y_r = x`_r \sin -\alpha + y`_r \cos -\alpha$
  - $\alpha_r = \alpha_0 - \alpha$

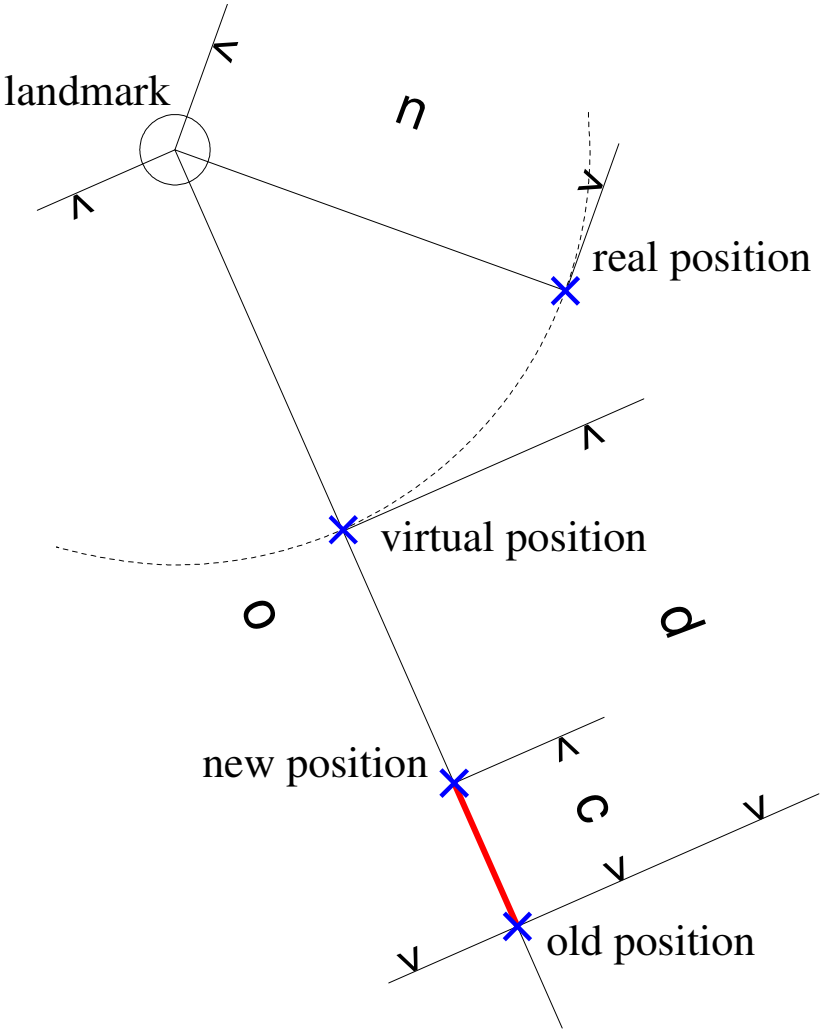


Localization – Odometry



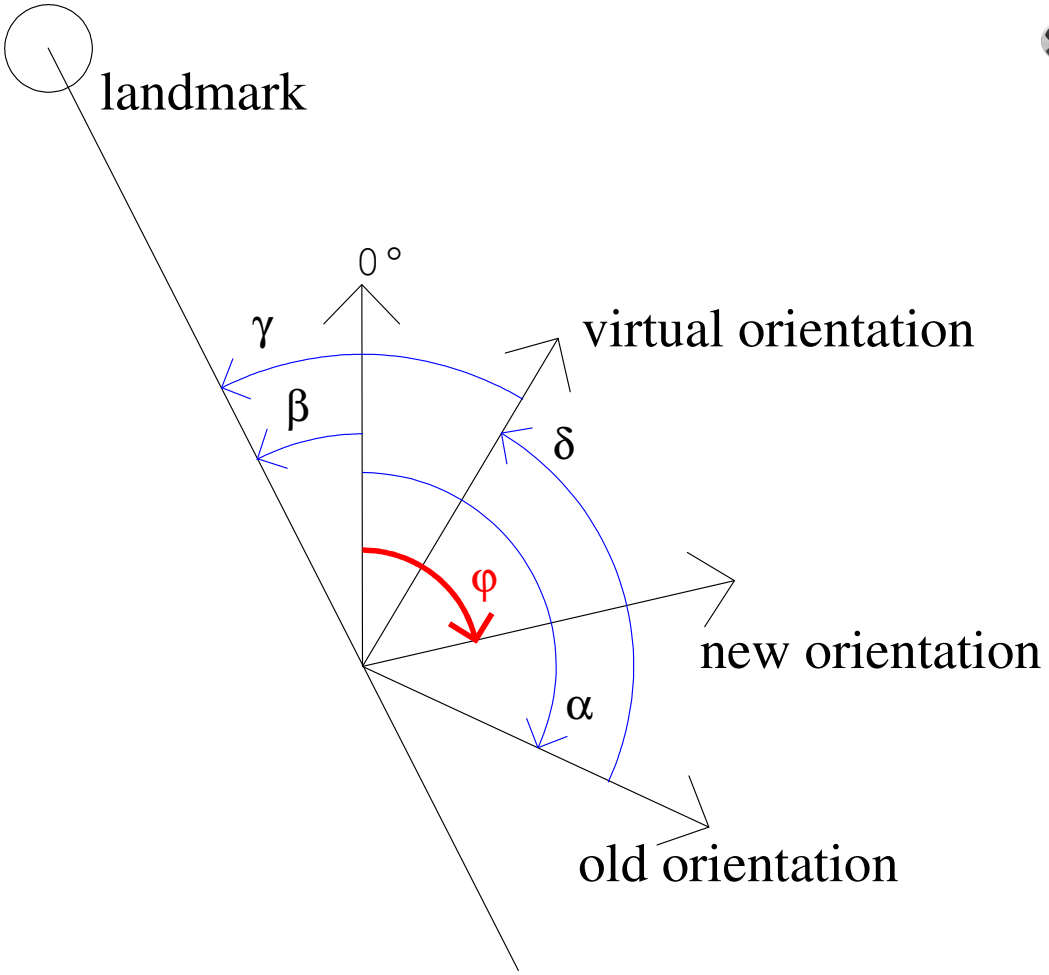


Localization – Iterative Approach



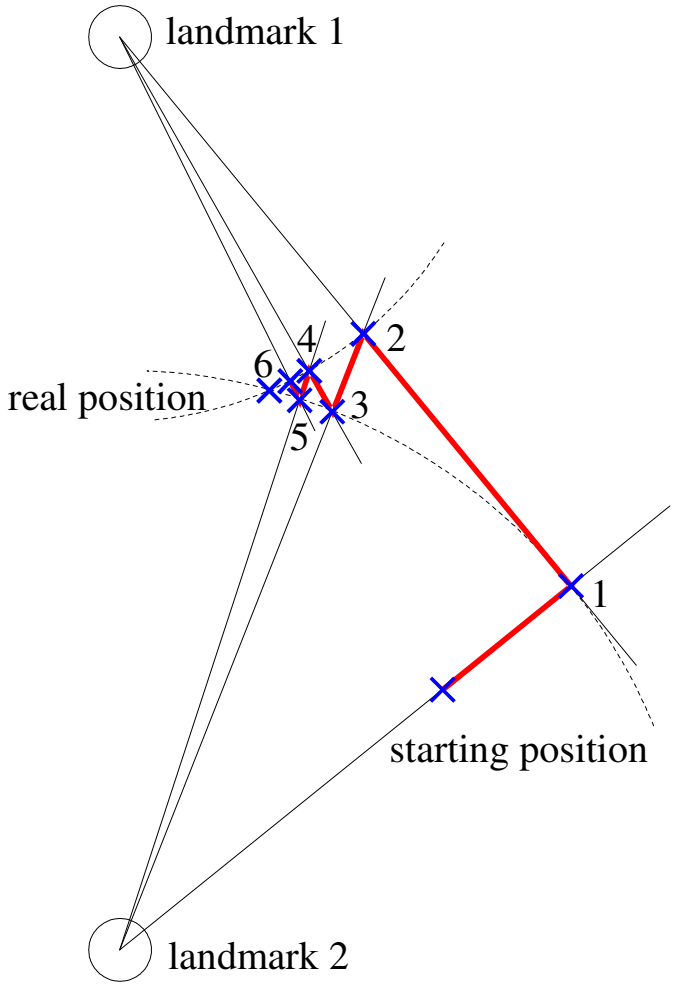
- Calculation of new position based on
  - old position
  - Reliability of old position
  - current distance measurement **n**
  - Reliability of the measurement
- New position is always on the straight line between old position and landmark

Localization – Iterative Approach



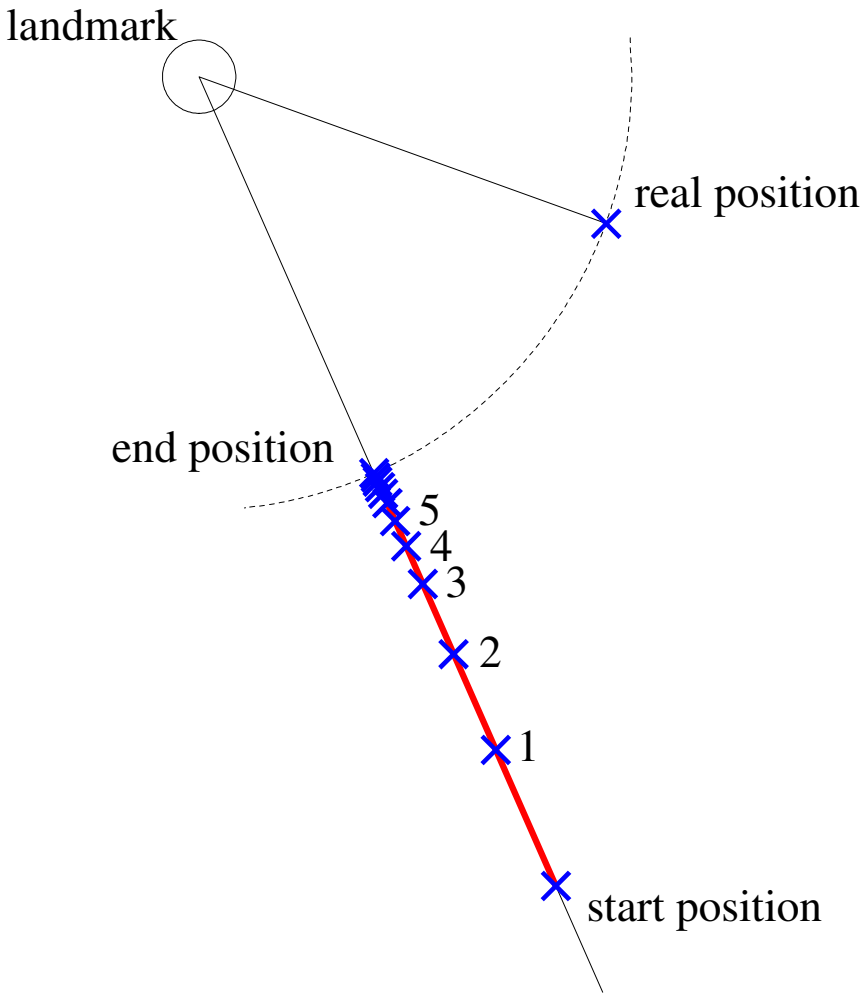
- Calculation of new orientation φ based on
  - old orientation α
  - Reliability of the old orientation
  - Measured angle γ to landmark
  - Reliability of the measurement

Localization – Iterative Approach



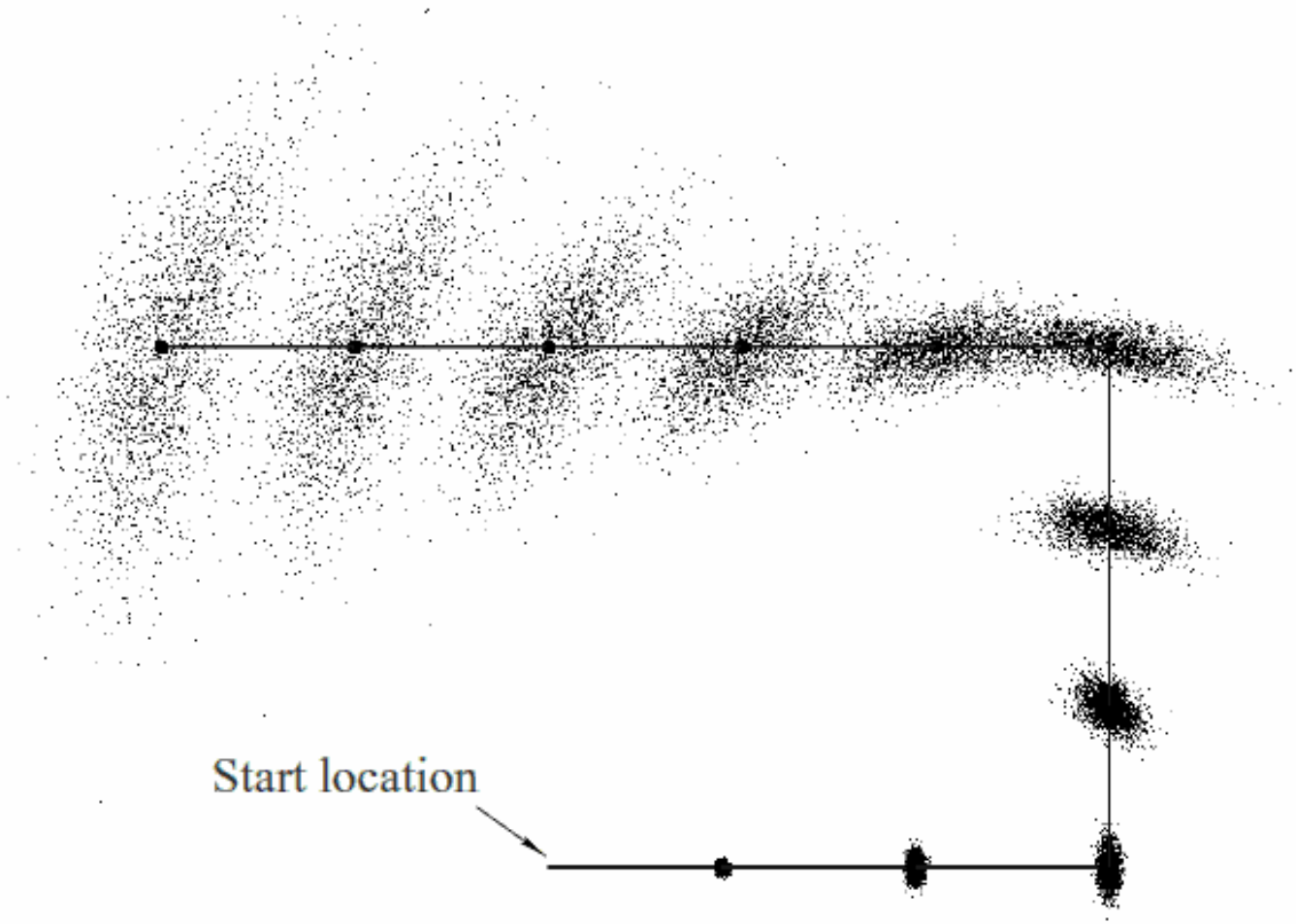
- Normal case
  - Different landmarks are seen alternately
- Result
  - The estimated position converges towards the real position

Localization – Iterative Approach

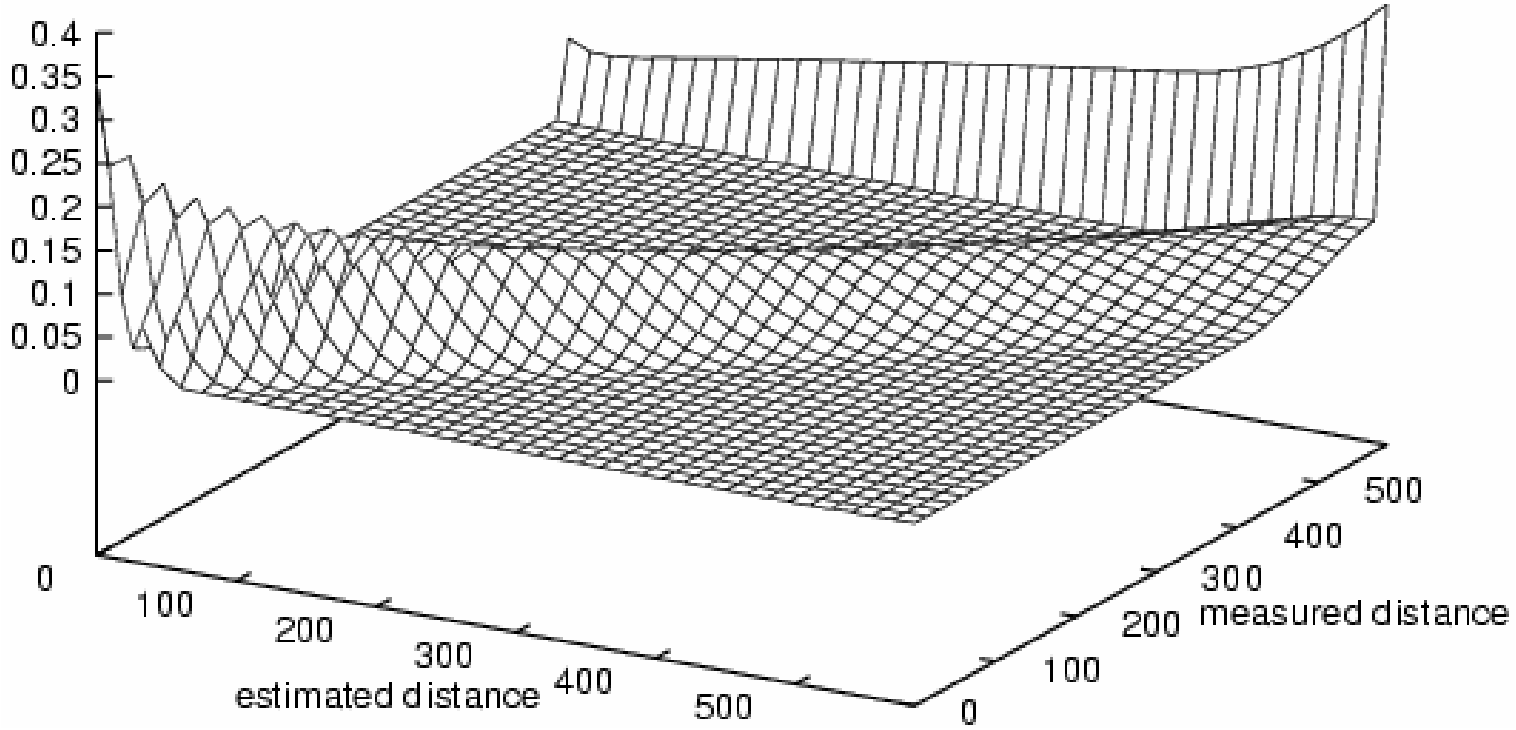


- Problem
  - Only a single landmark is visible for a long time
- Result
  - Position is always on the straight line between the initial position and the landmark

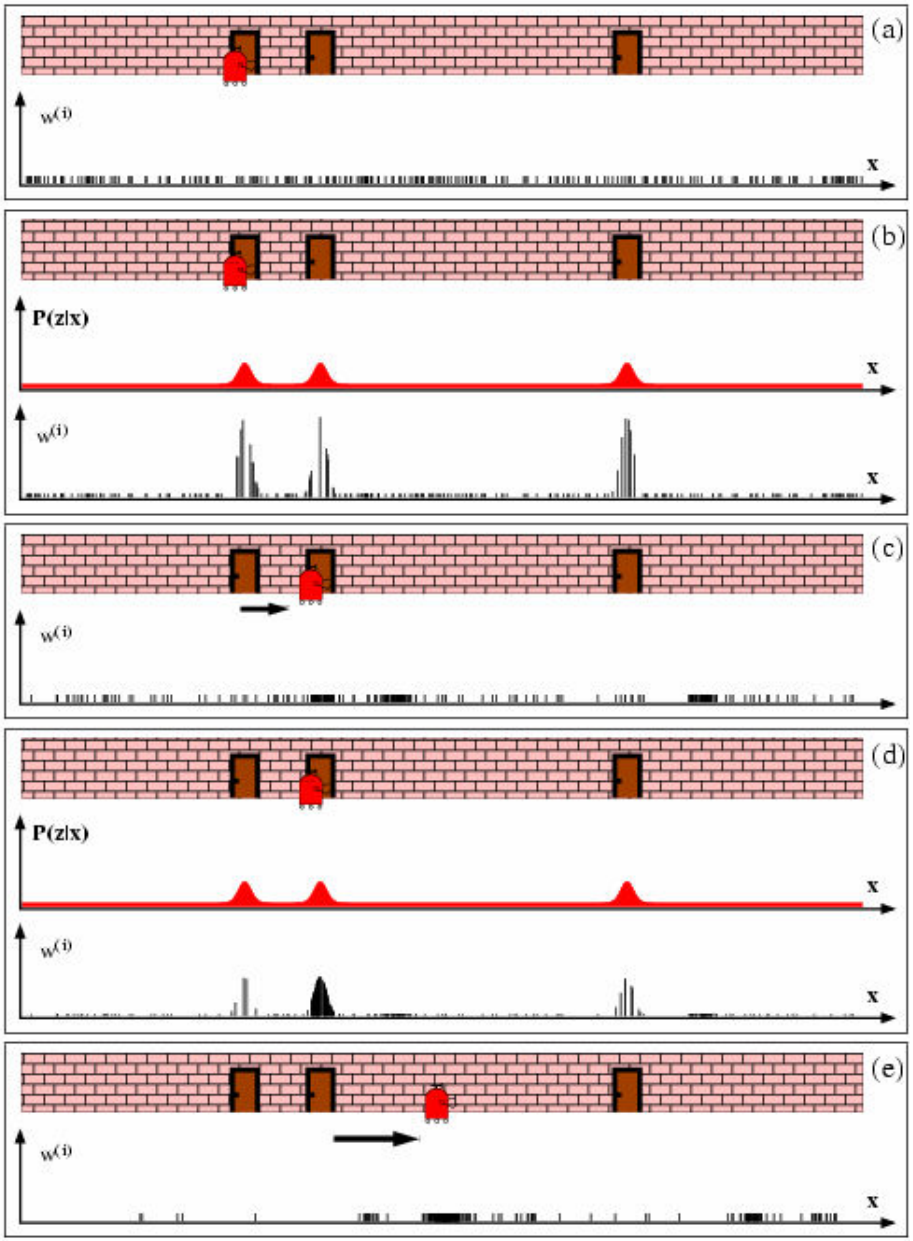
Localization – Uncertainty in Motion

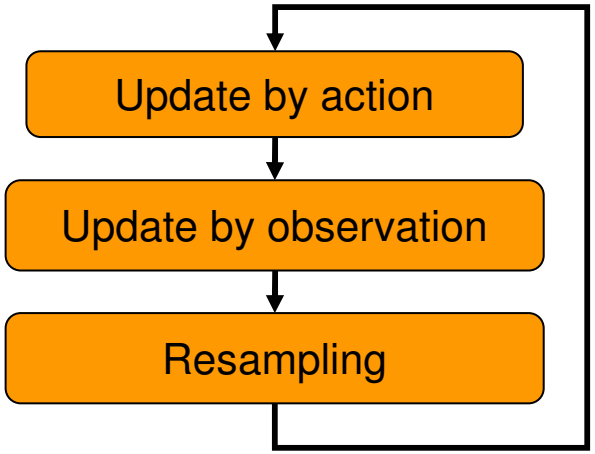


Localization – Uncertainty in Observation

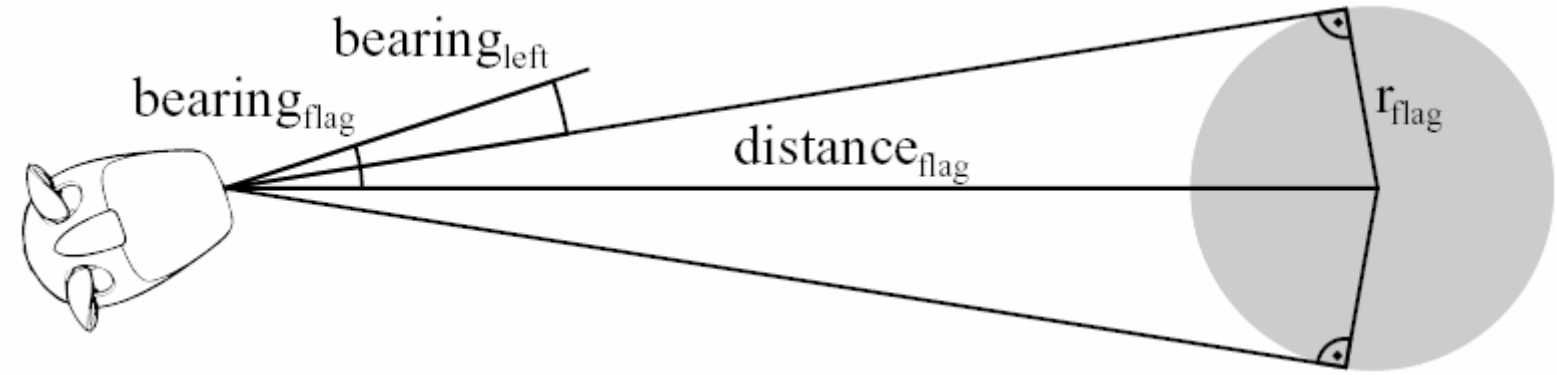


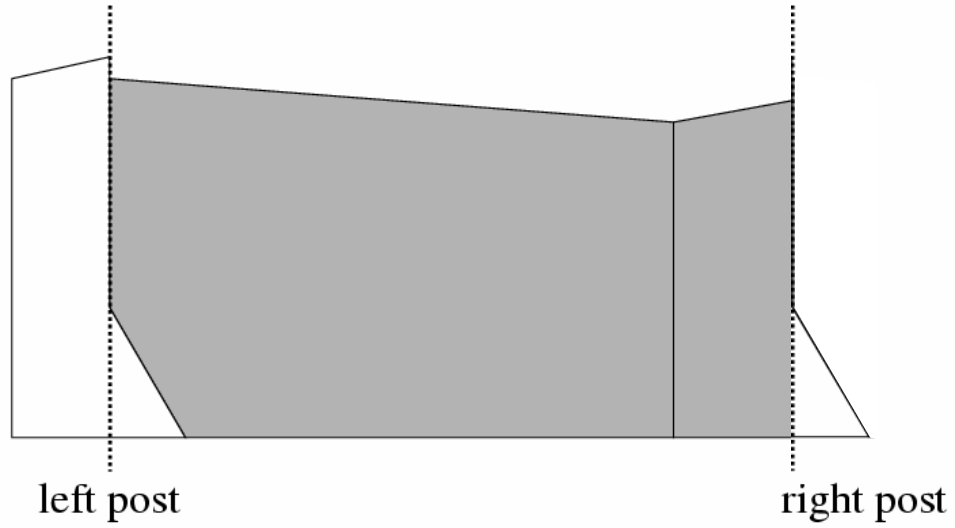
Localization – General Approach



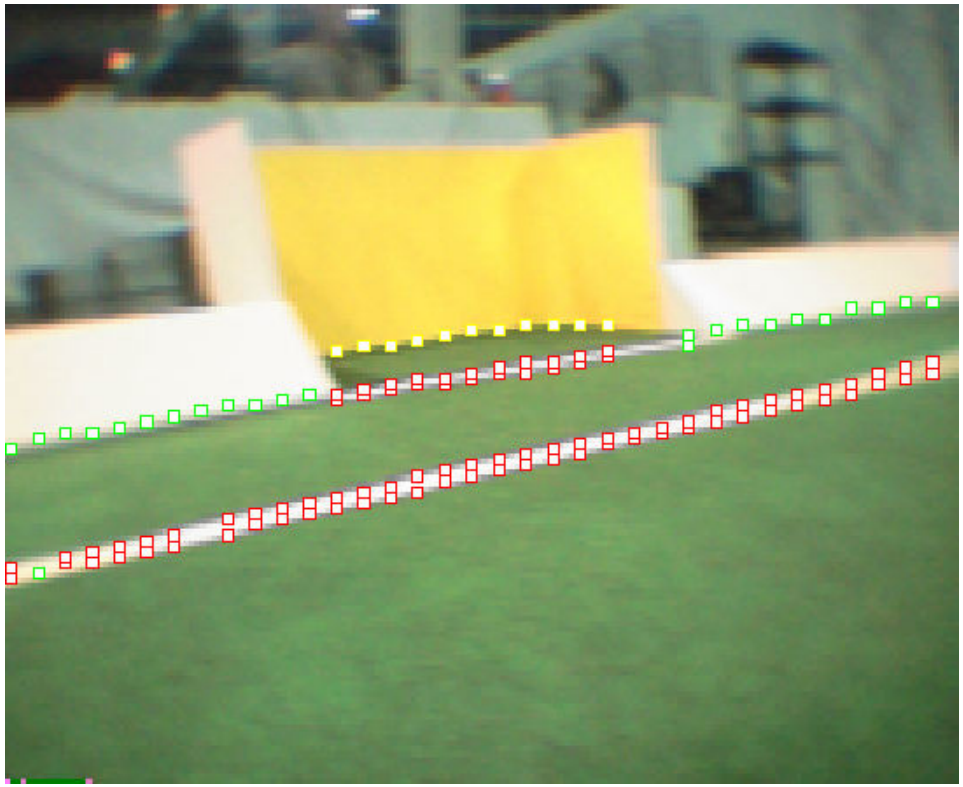


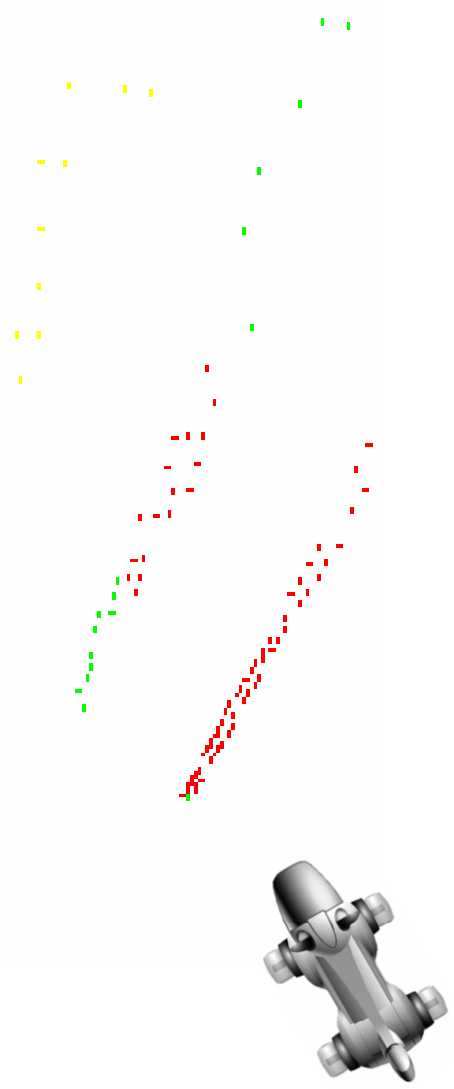
Localization – Observations



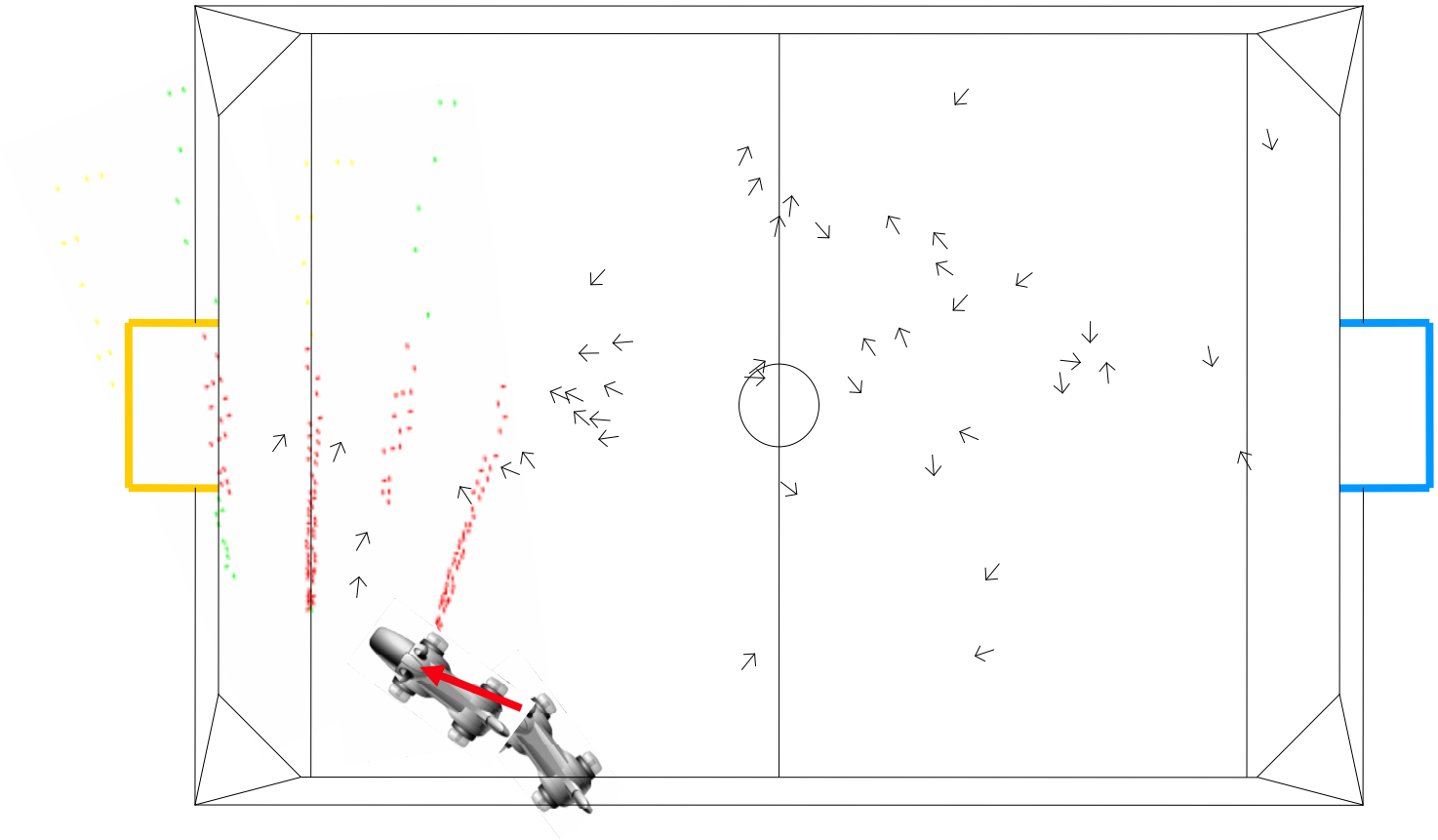


Localization – Observations

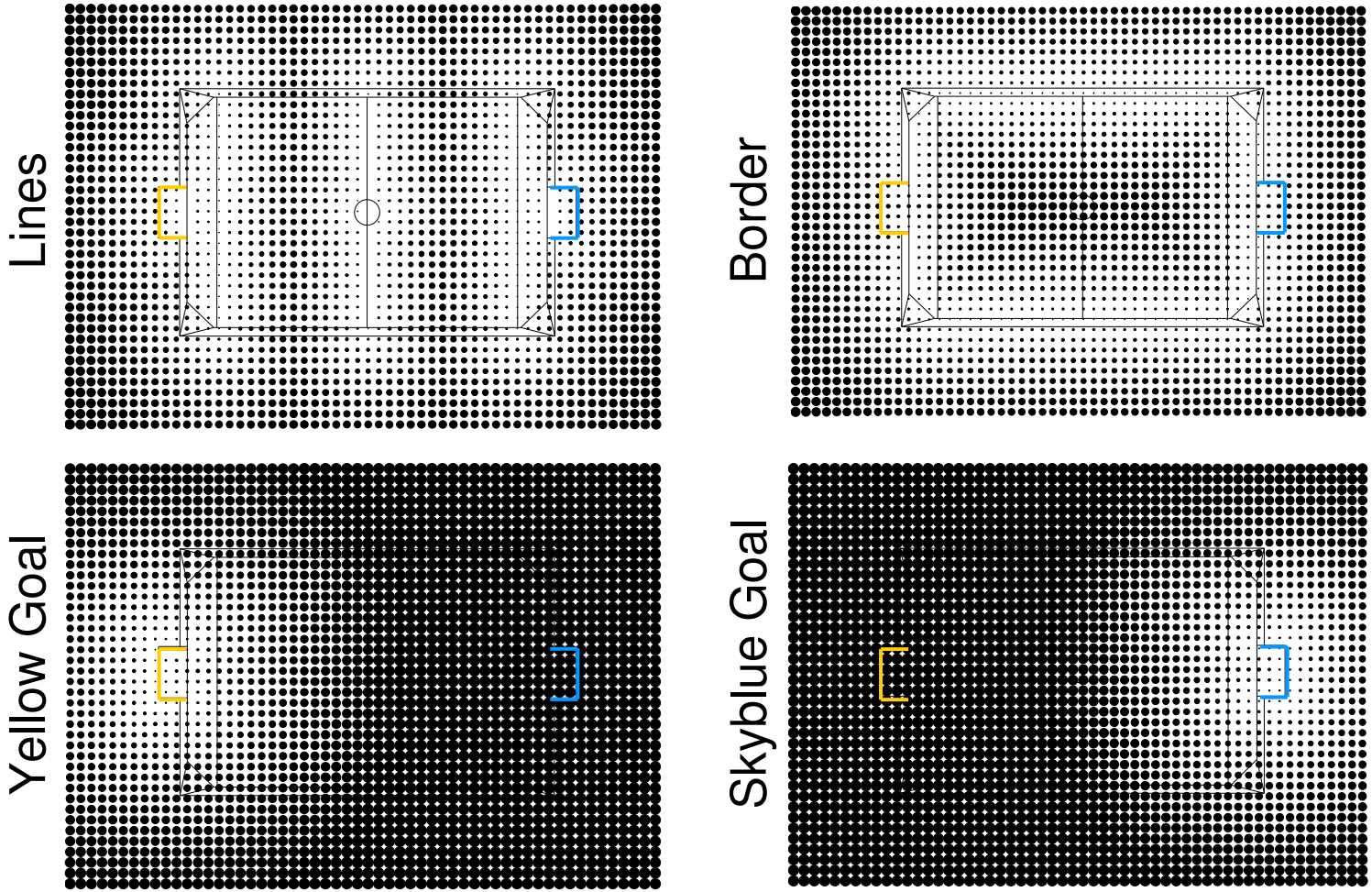




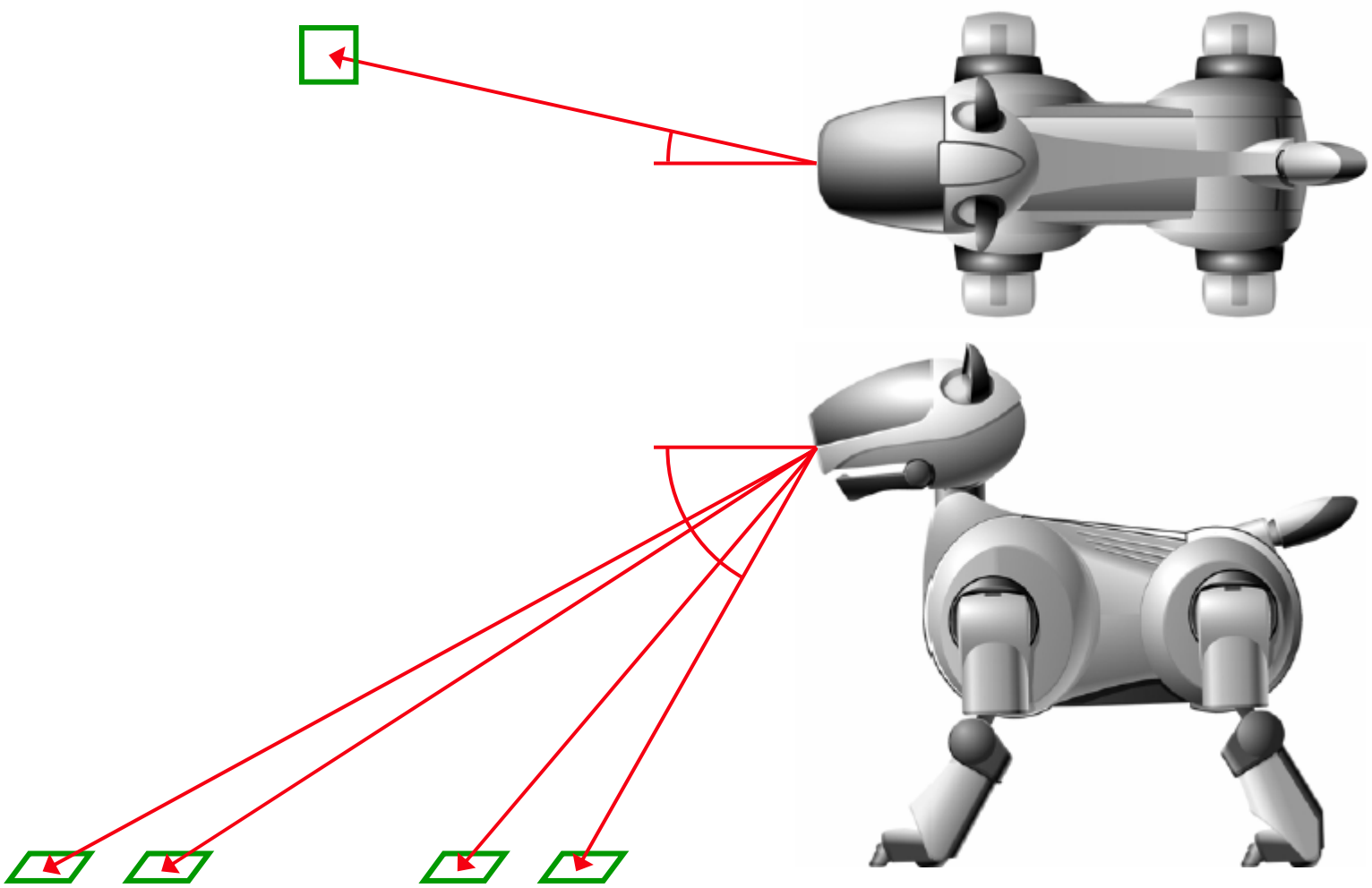
Localization – Approach



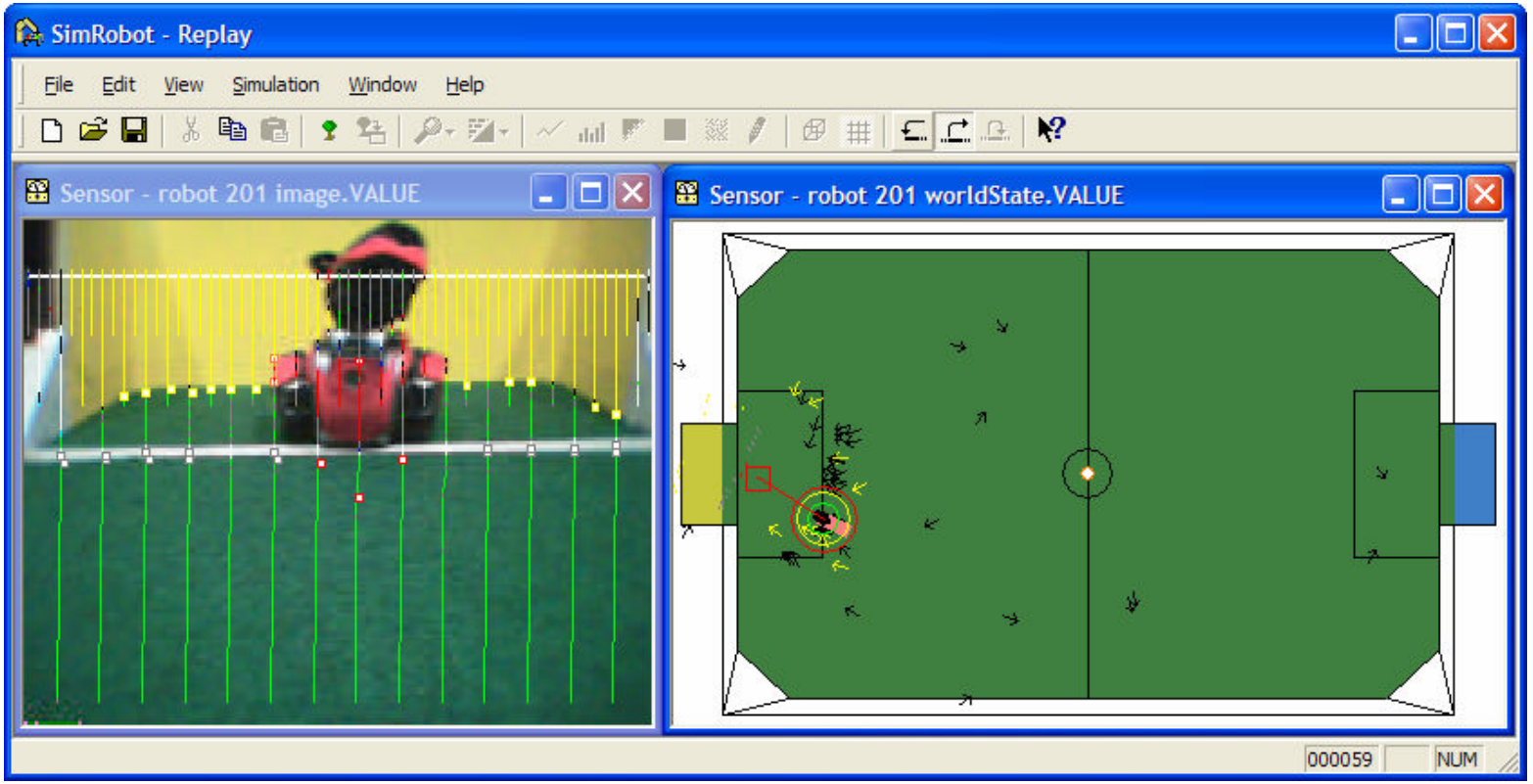
Localization – Assigning Observations to Field Model



Localization – Sensor Model



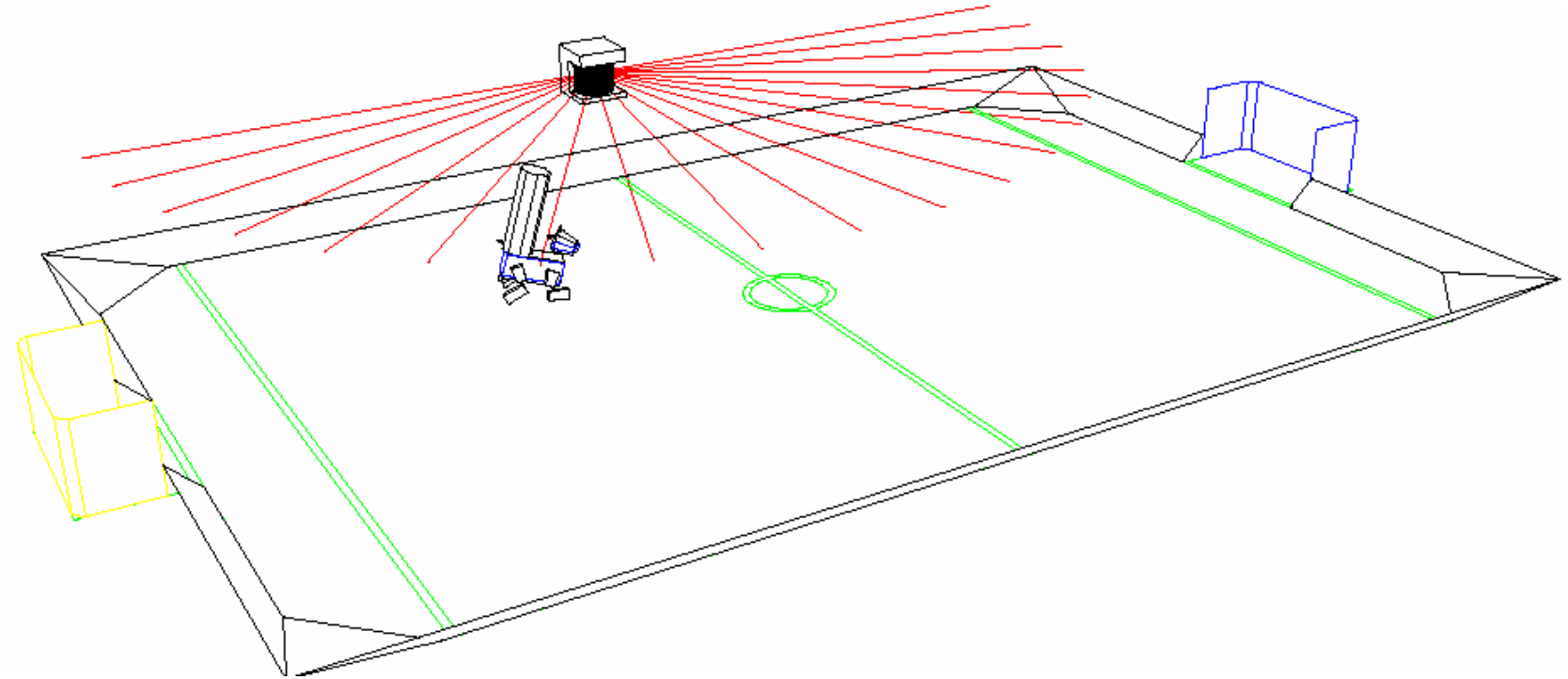
Localization – Example



Localization – Details

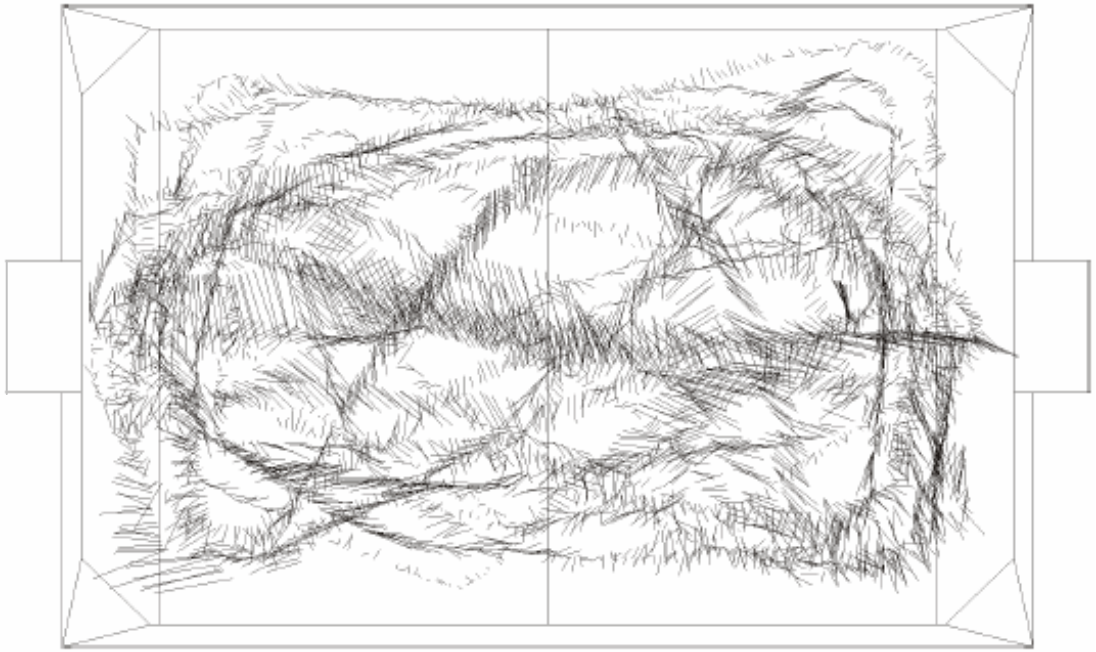
- Probability of samples
  - Probability is adapted slowly
  - Separate probabilities for different observation types
  - Samples are randomly moved, weighted by their probabilities
- Sensor resetting
  - Draw samples based on the ratio of their probability and the average probability
  - Replace them by candidate postures that can be derived from observations
- Calculating candidates in advance
  - A large number of random postures is generated
  - Their distance to the edge they are pointing to is determined
  - The postures are indexed by their distance and edge type

Localization – Experimental Setup



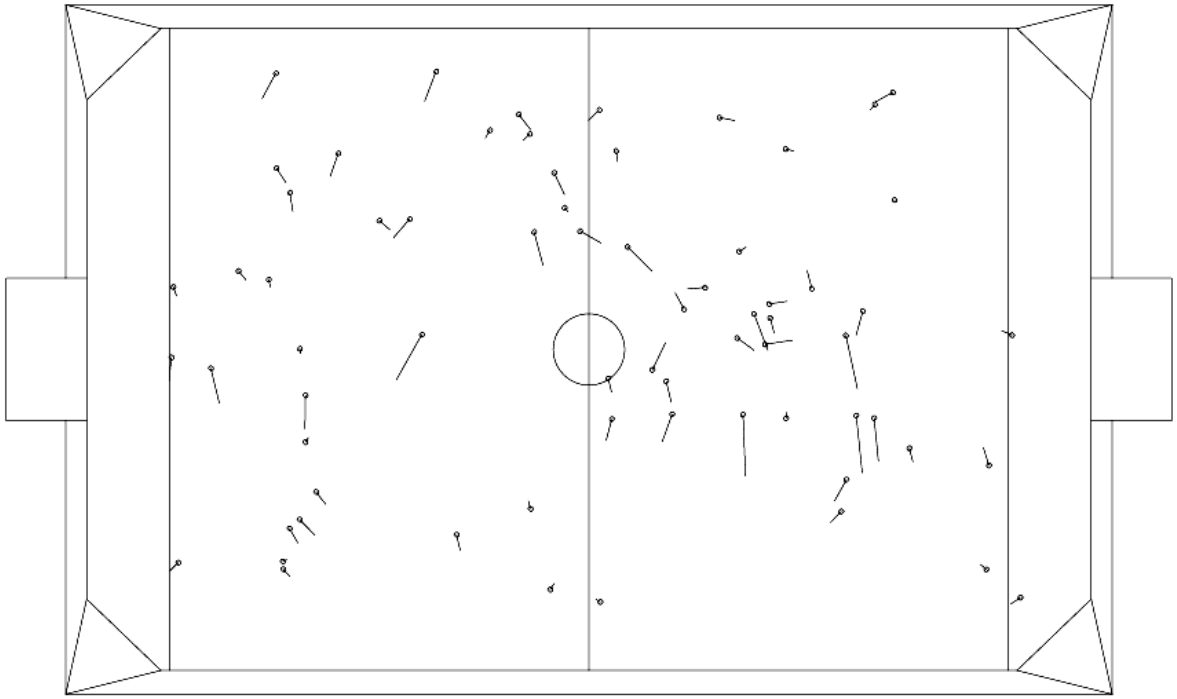
Localization – Experiment 1

- Robot continuously moving (by joystick)
- Approx. 5300 measurements
- Average error < 10.5 cm
  (field size is 420 x 270 cm²)



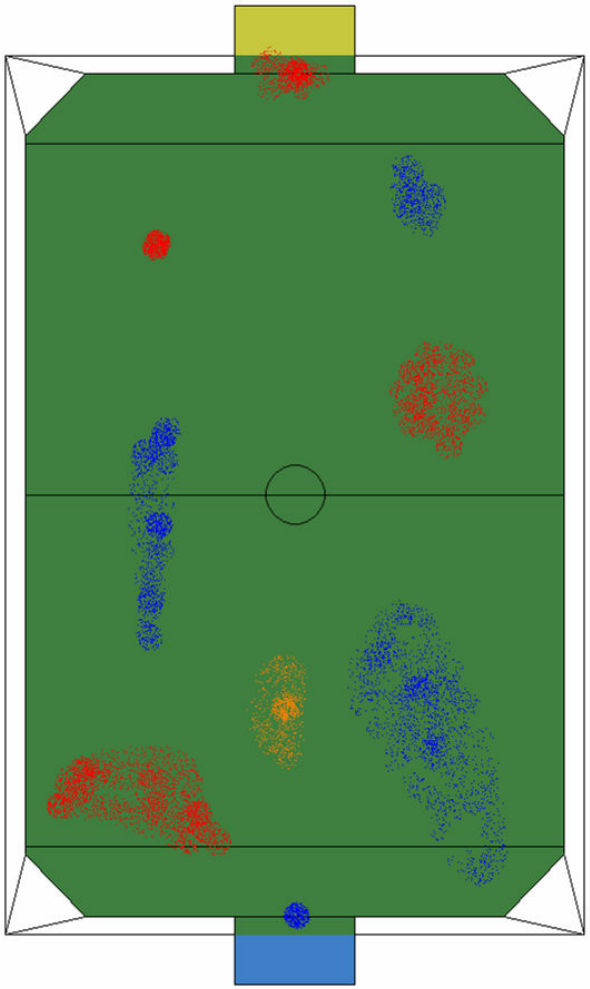
Localization – Experiment 2

- Robot walks to random positions (approx. 70)
- Average error in positioning < 9.5 cm
- Average error in localization < 8.5 cm

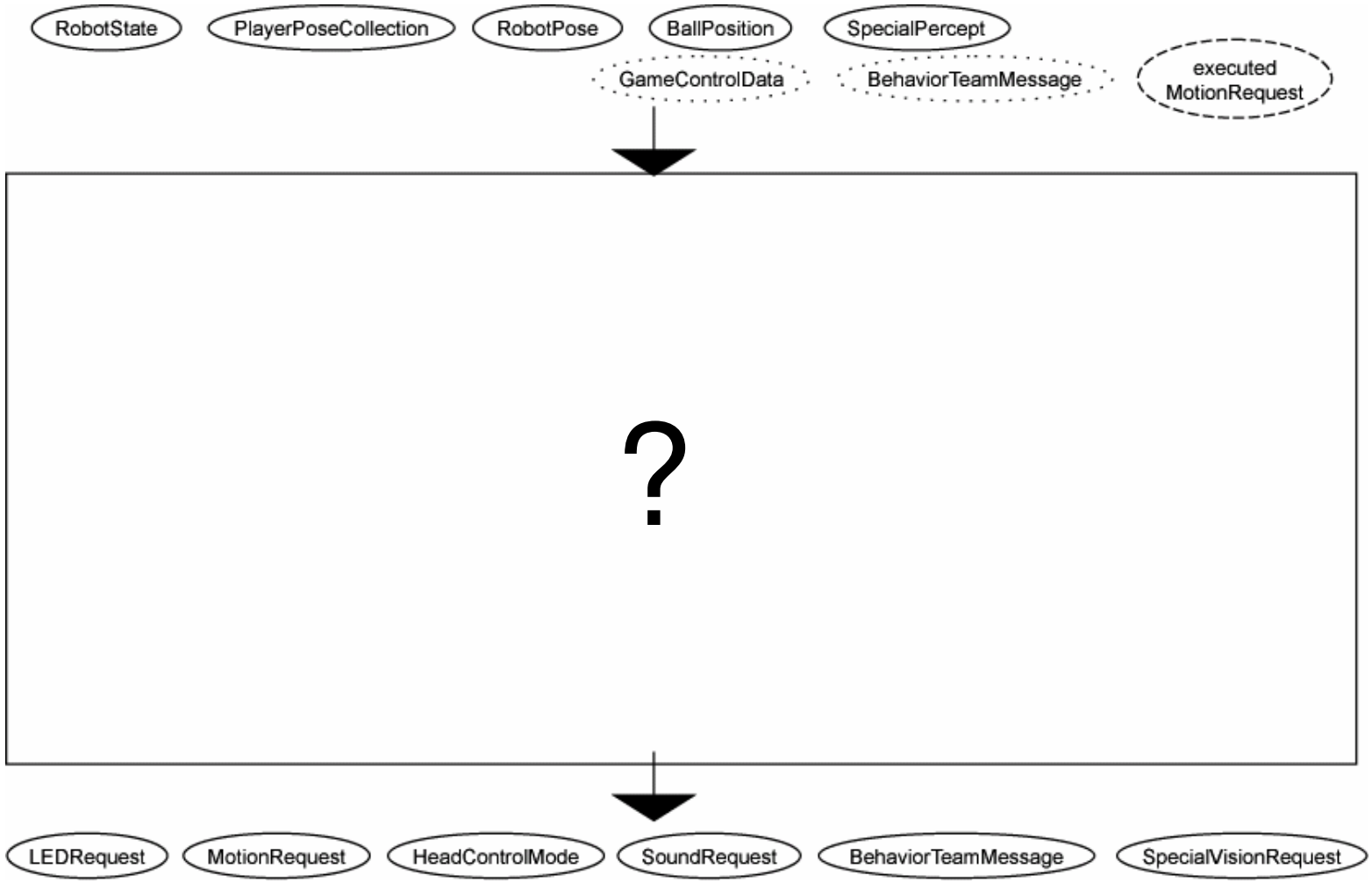


World Modeling - Probabilistic Approach

- Modeling the
  - own position
  - position of the ball
  - positions of teammates
  - positions of opponents
  - positions of obstacles
- Hierarchy
  - Local world model
  - Shared world model
- Communication as essential part



Behavior Control

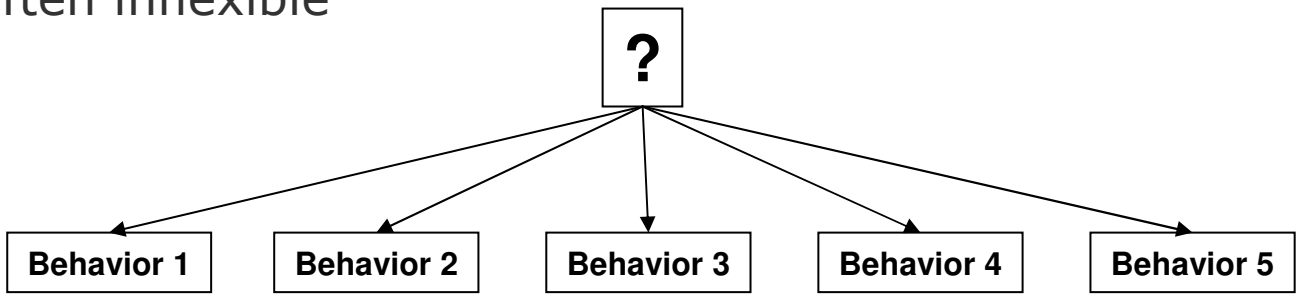


Behavior Control – Demands

- Uncertainty: noisy world model
- Communication with teammates: long delays
- Flexibility vs. stability
  - Flexibility:
    - *Quick changes of behavior / adaptation to the environment*
    - *Drawbacks: oscillations, behaviors are not finished*
  - Stability:
    - *Continuing behaviors if they were started*
    - *Advantages: Reliability during cooperation, better handling of imprecise perception. Long-term behaviors can be performed*
    - *Drawback:* fanaticism

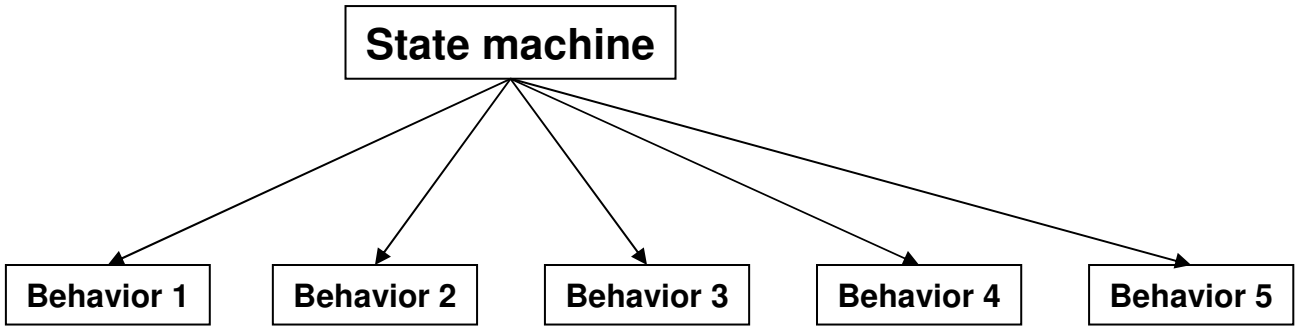
Behavior Control – Approaches

- Approaches
  - With utilities
  - Potential fields
  - Artificial Neural Networks
  - Decision trees
- Problems
  - Decision do not only depend on sensor data but also on the context
  - Universal hierarchies of utilities are hard to find
  - Hard to continue behaviors once started
  - Often inflexible

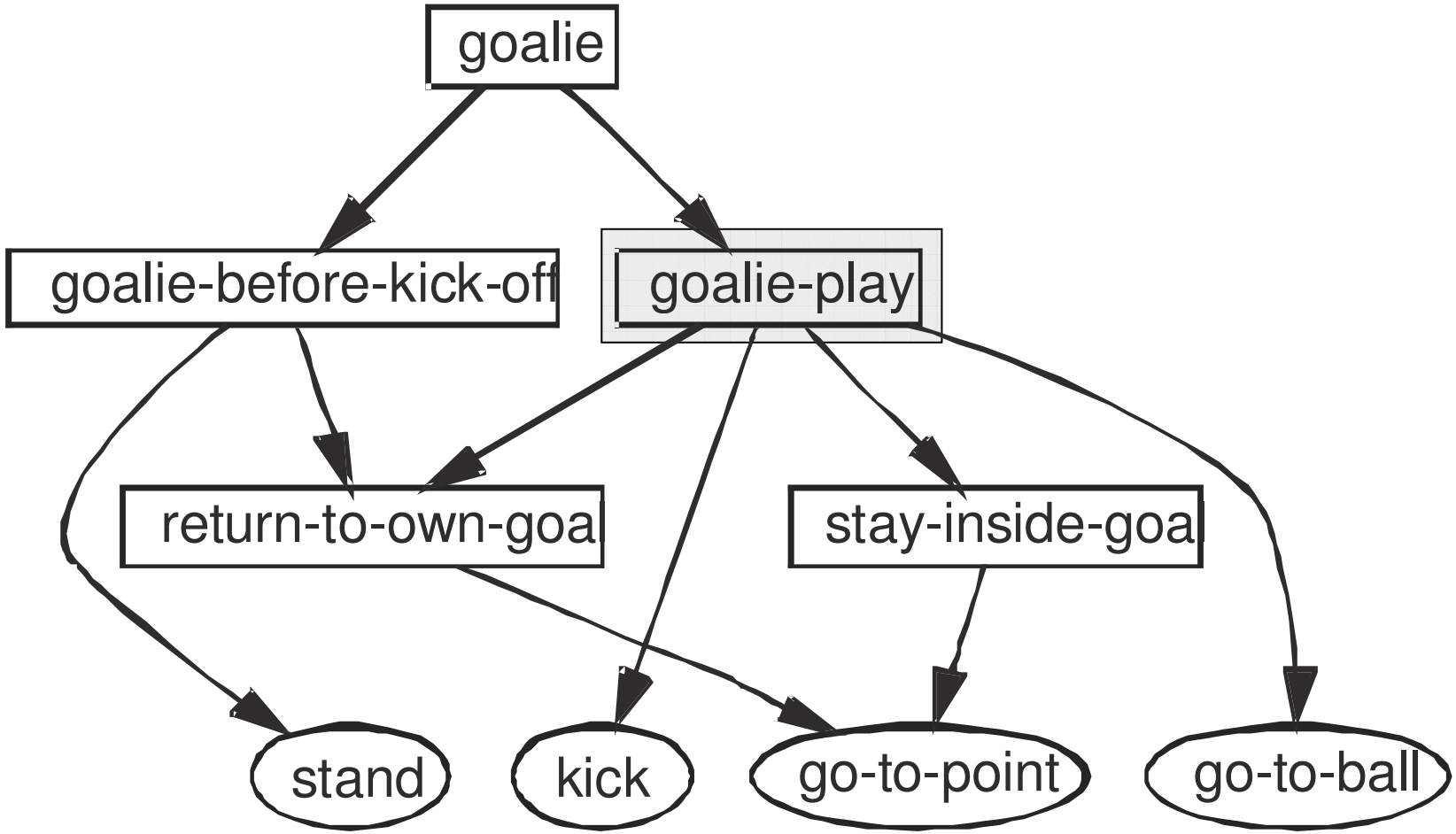


Behavior Control – State Machines

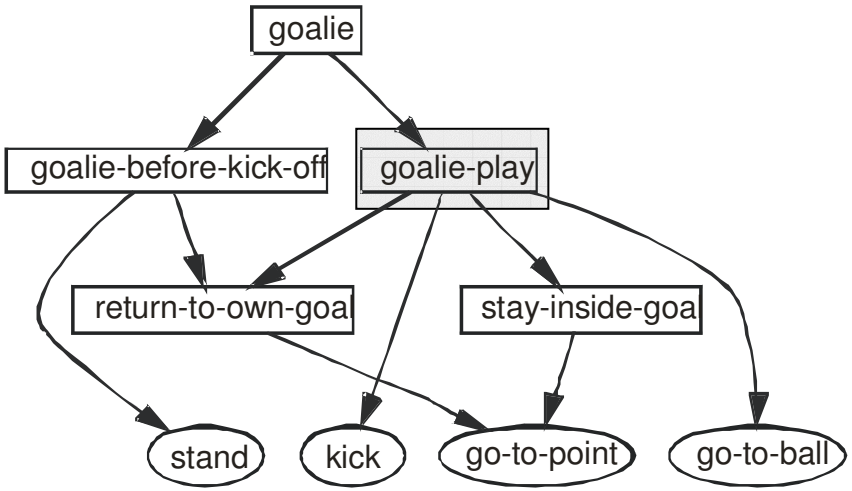
- Behavior selection with state machines
  - Currently selected behavior corresponds to the state of a state machine
  - A state remains actual until a condition for switching to a different state is satisfied
  - Hence: only sequences of sensible behaviors
  - Hysteresis is possible
- Context problem
  - The transitions between states differ in different contexts → **Hierarchy of state machines**

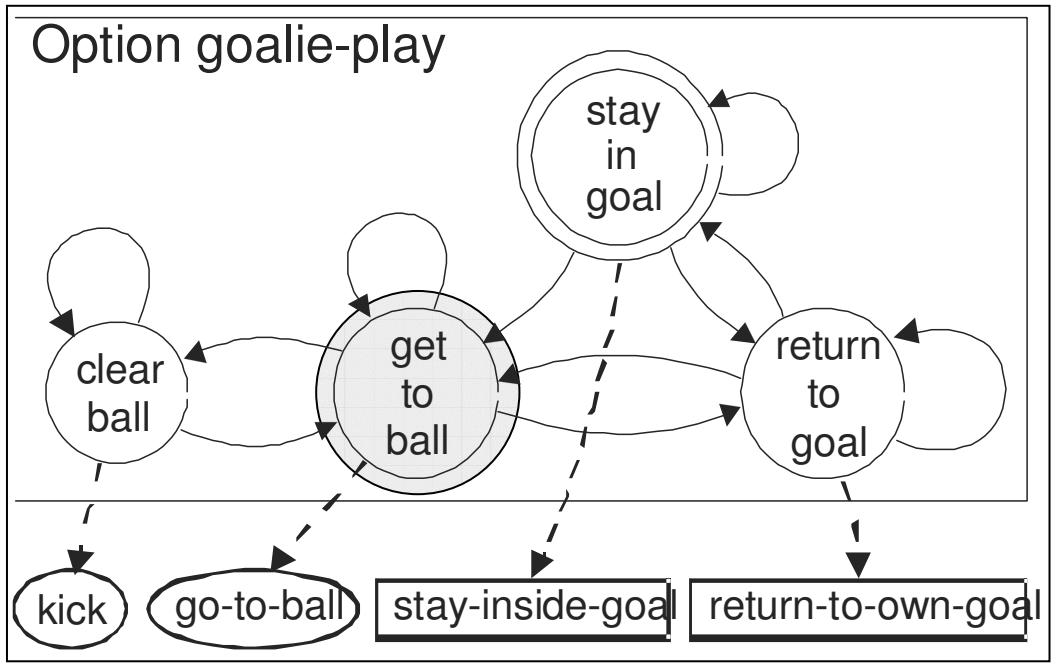


Behavior Control – Hierarchy of Options

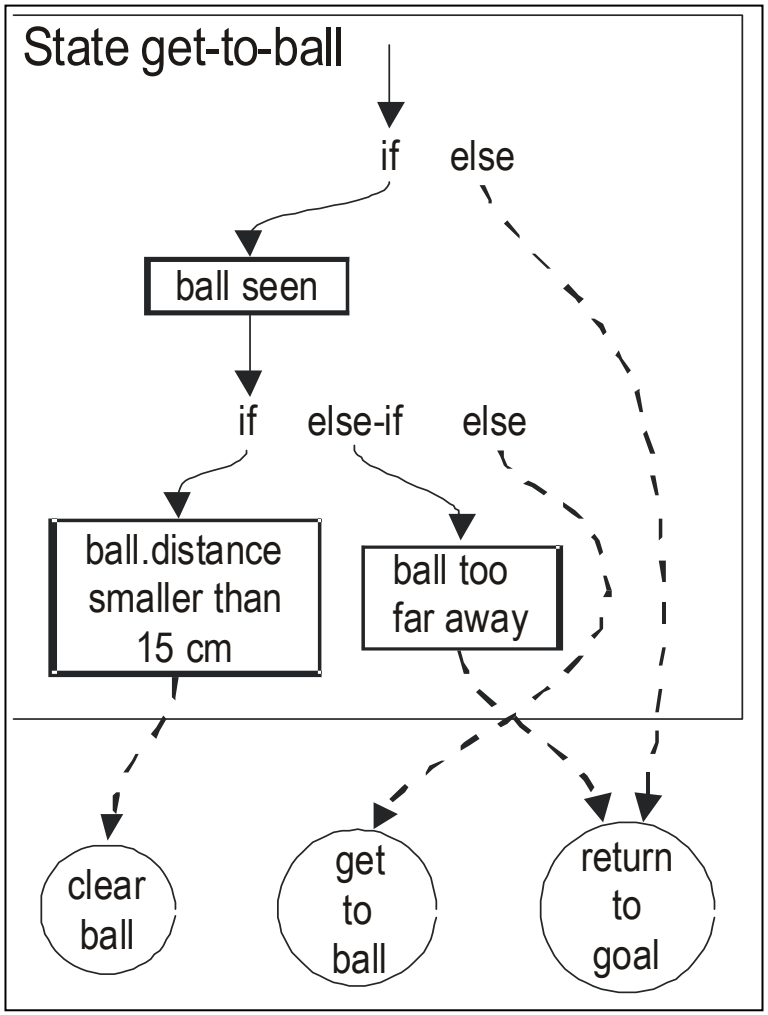


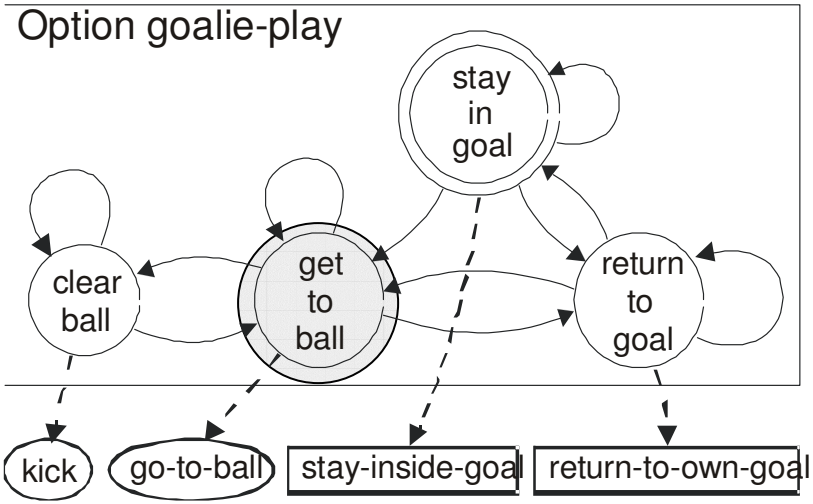
Behavior Control – Options → States



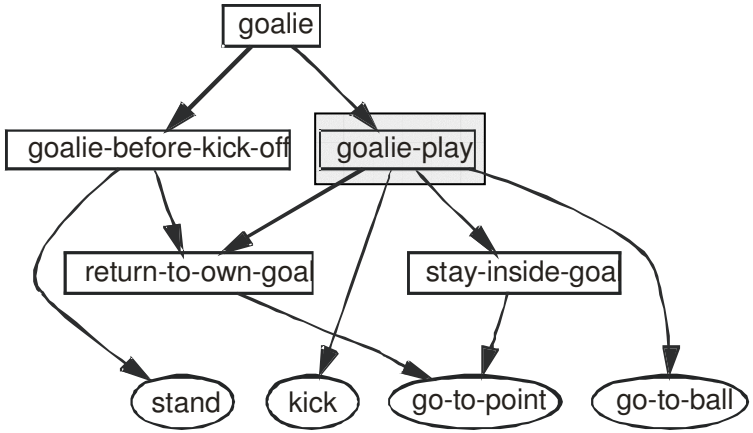


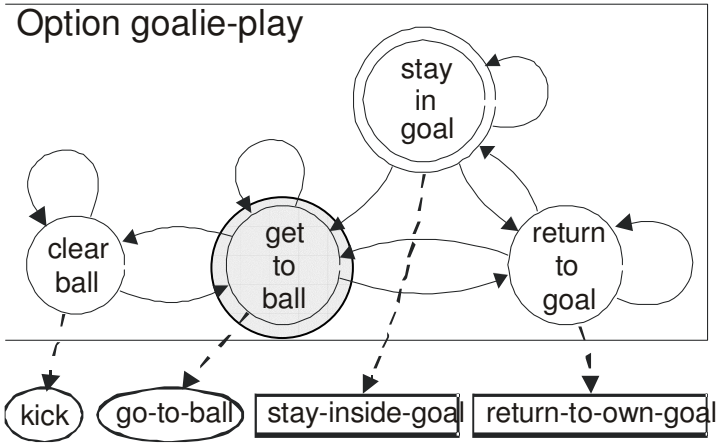
Behavior Control – States → Decision Trees

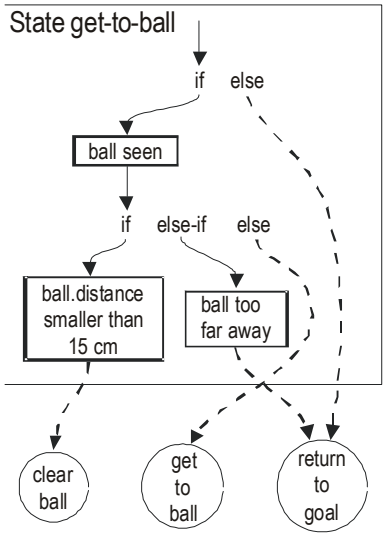




Behavior Control – Hierarchy of Options

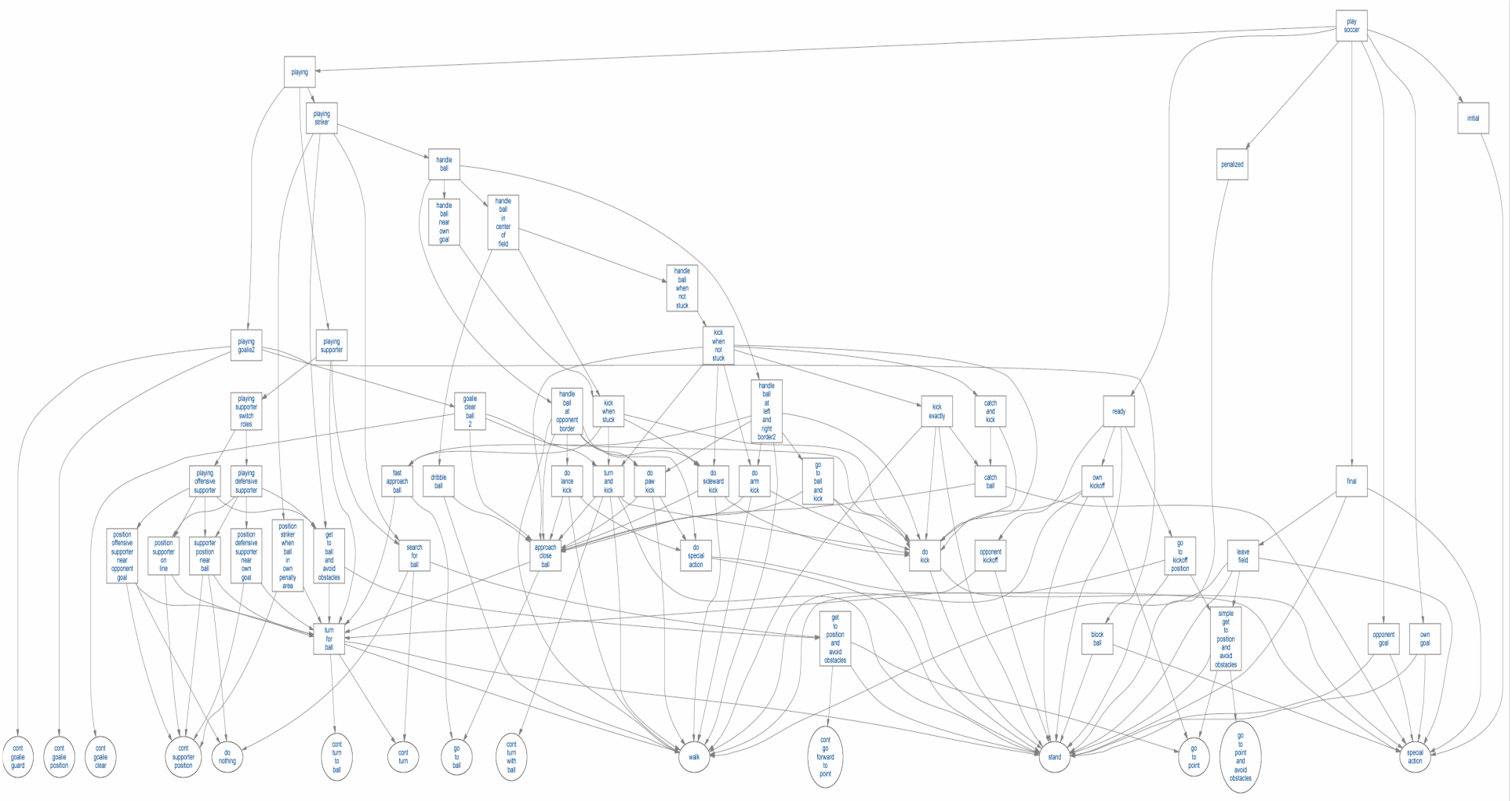







```
if  (ball.time-since-last-seen < 2000)                    //ball  seen
{
  if   (ball.distance < 150)  //ball.  distance  smaller  than  15  cm
  {
    transition-to-state (clear-ball);
  }
  else  if   (ball.distance > 900)            //bal  l too far    away
  {
    transition-to-state (return-to-goal);
  }
  else
  {
    transition-to-own-state (get-to-ball);
  }
}
else
{
 transition-to-state (return-to-goal);
}
```

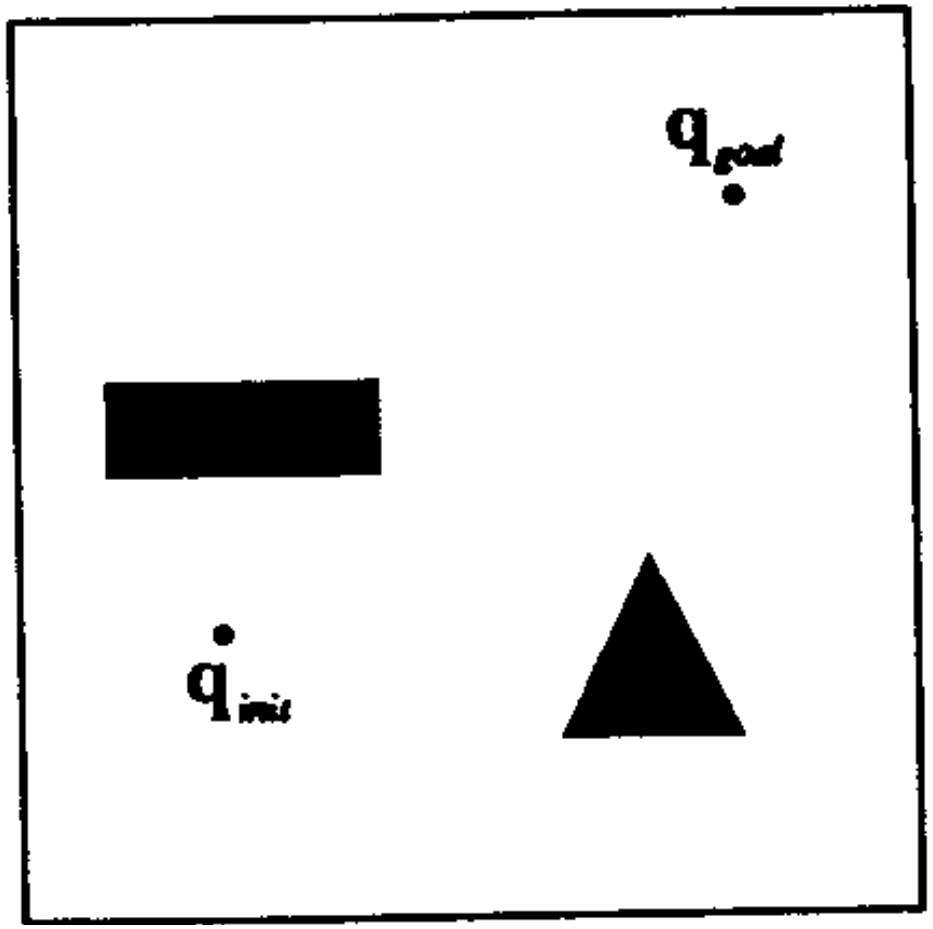
Behavior Control - Option Graph of GermanTeam 2003



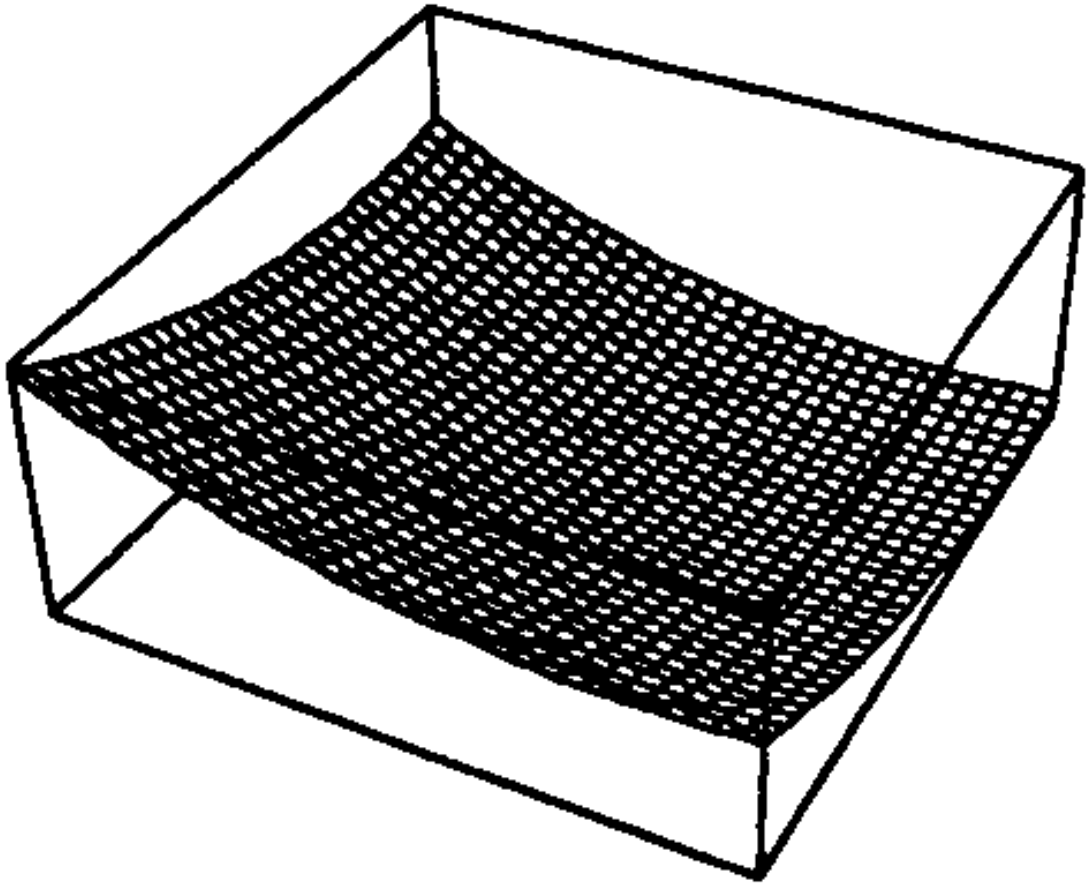
Behavior Control – Extensible Agent Behavior Specification Language (XABSL)

- Formalization of the software environment
  - Definition of basic behaviors and symbols
- Formalization of the option tree
  - Options
  - States
  - Decision trees
  - Conditions
  - Output Symbols
- Generation of
  - Intermediate code
  - Documentation
  - Debugging symbols
- XabslEngine
  - Executes intermediate code
  - Embedding into software environment
    - *XabslSymbolProvider*
    - *XabslBasicBehaviors*
  - Debugging interfaces

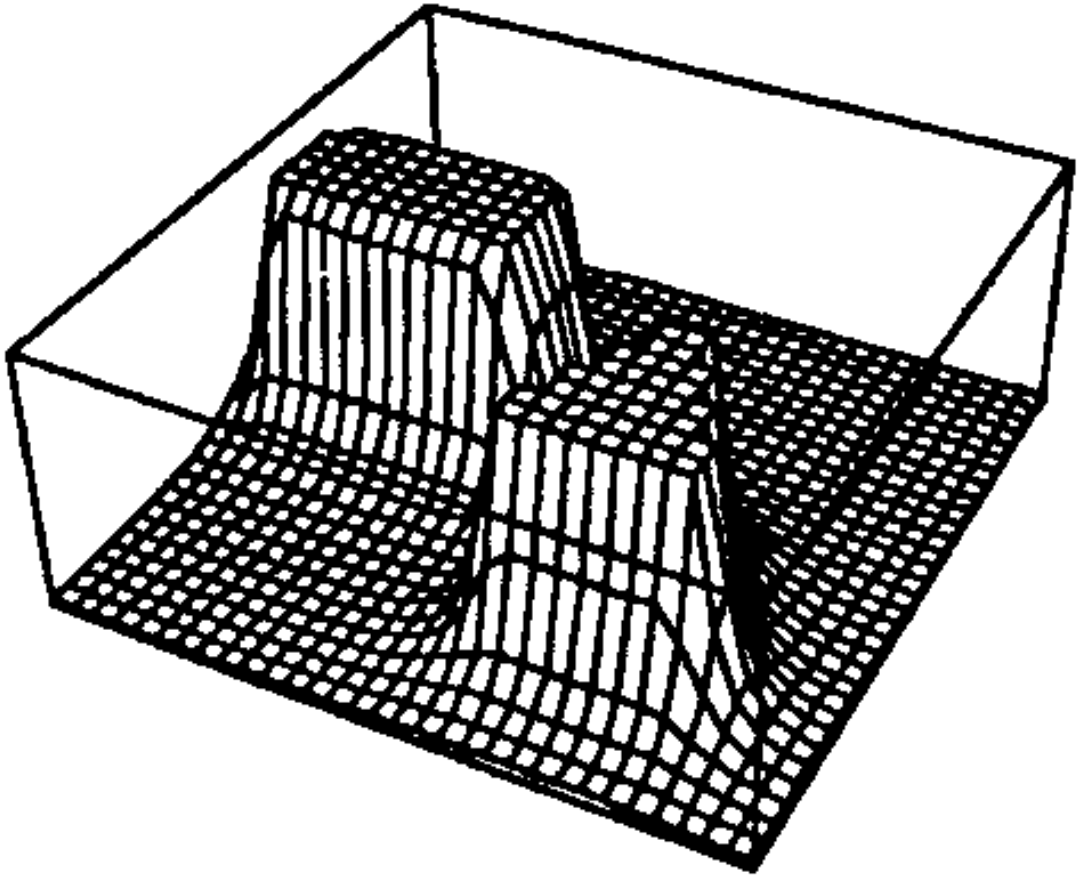
Potential Fields – Motion Planning



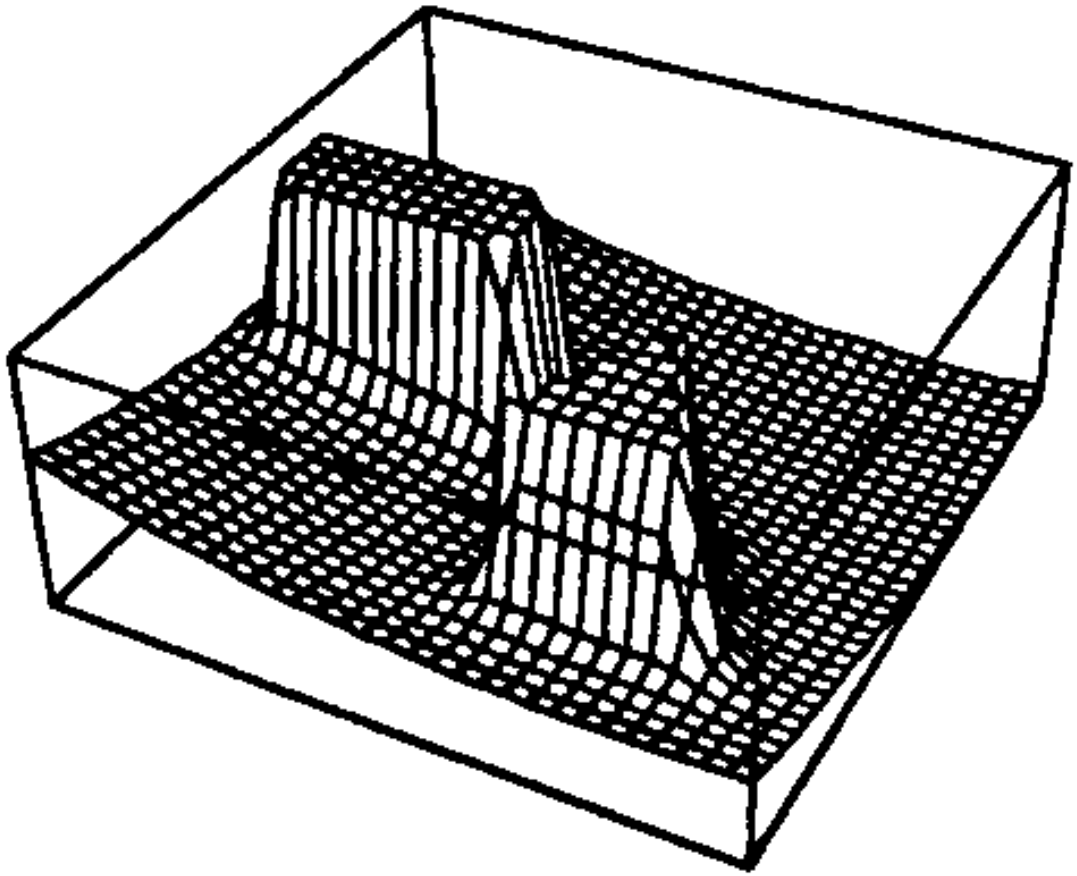
Potential Fields – Goal



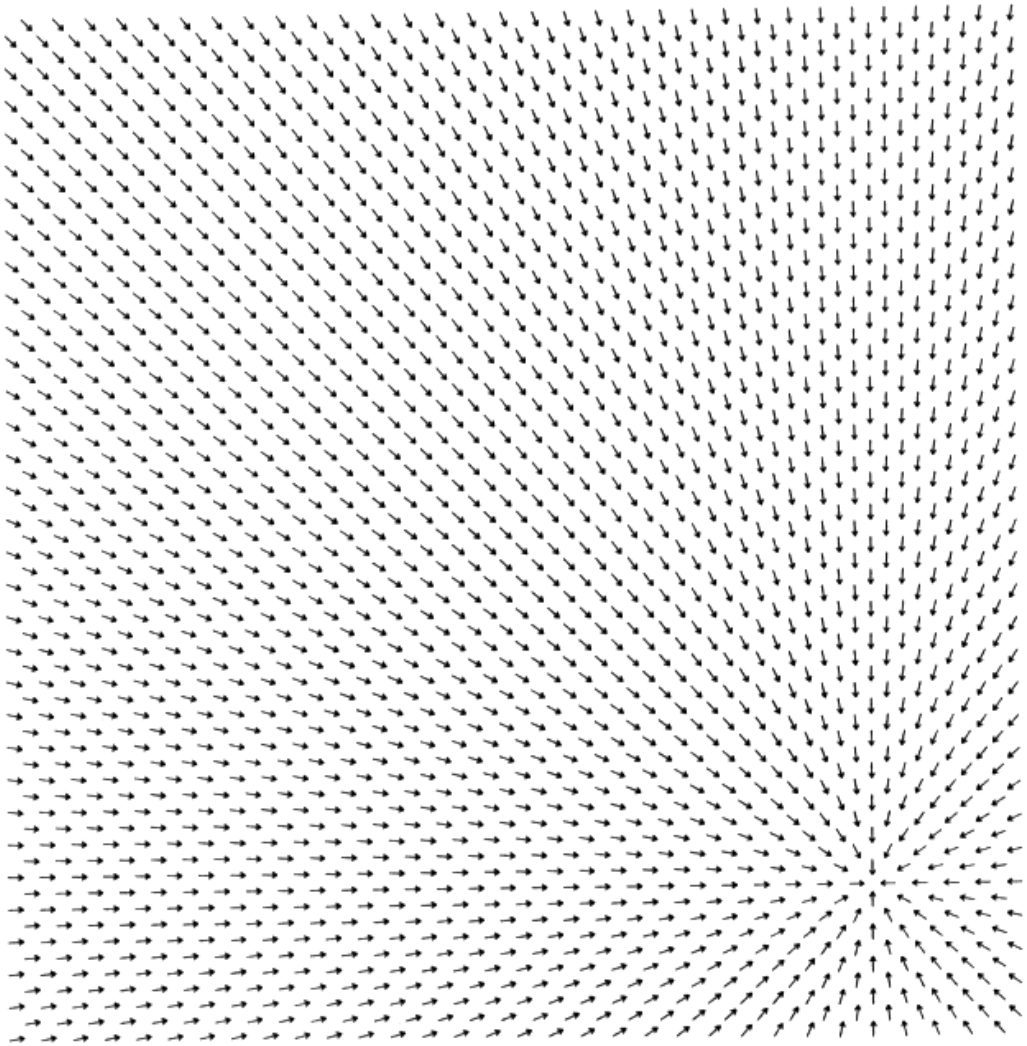
Potential Fields – Obstacles



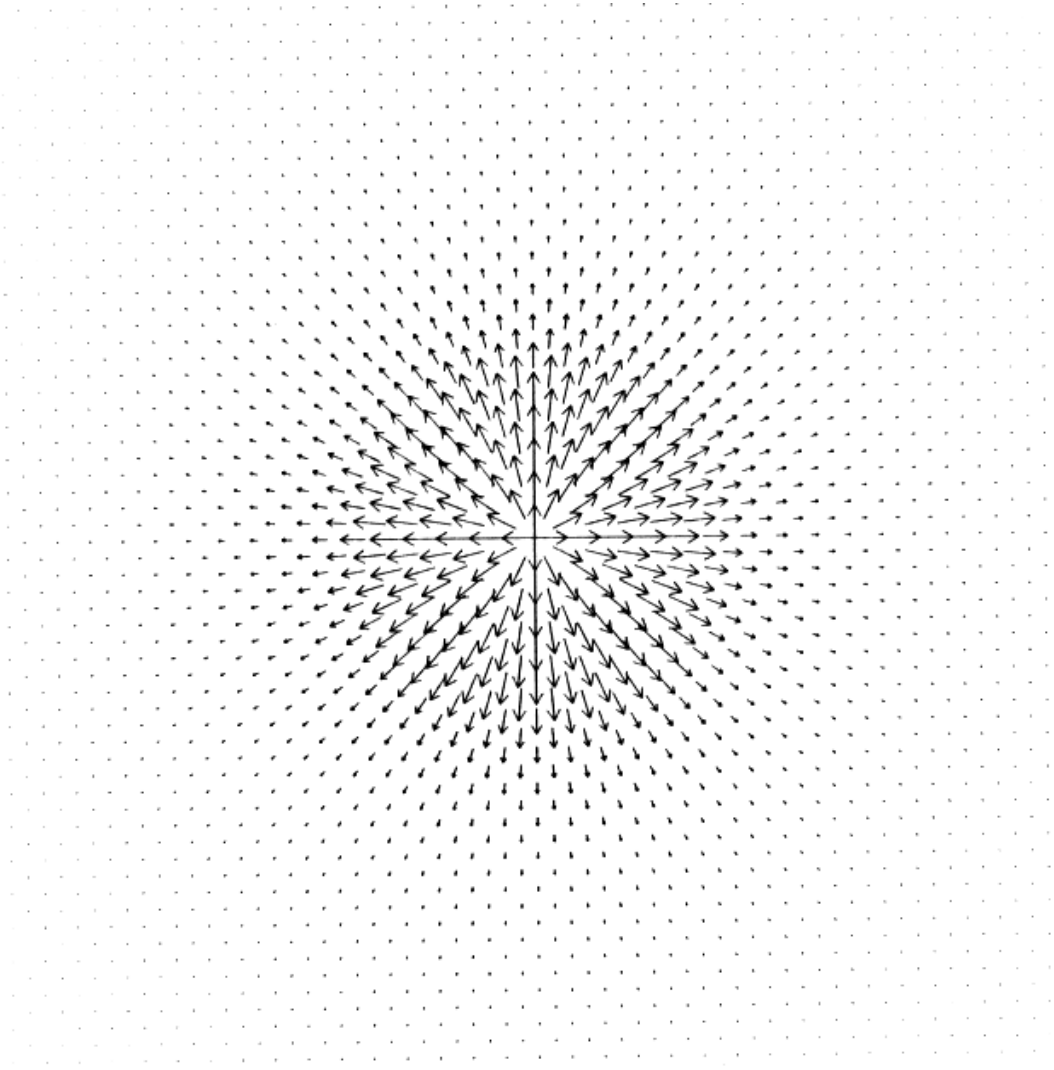
Potential Fields – Combination



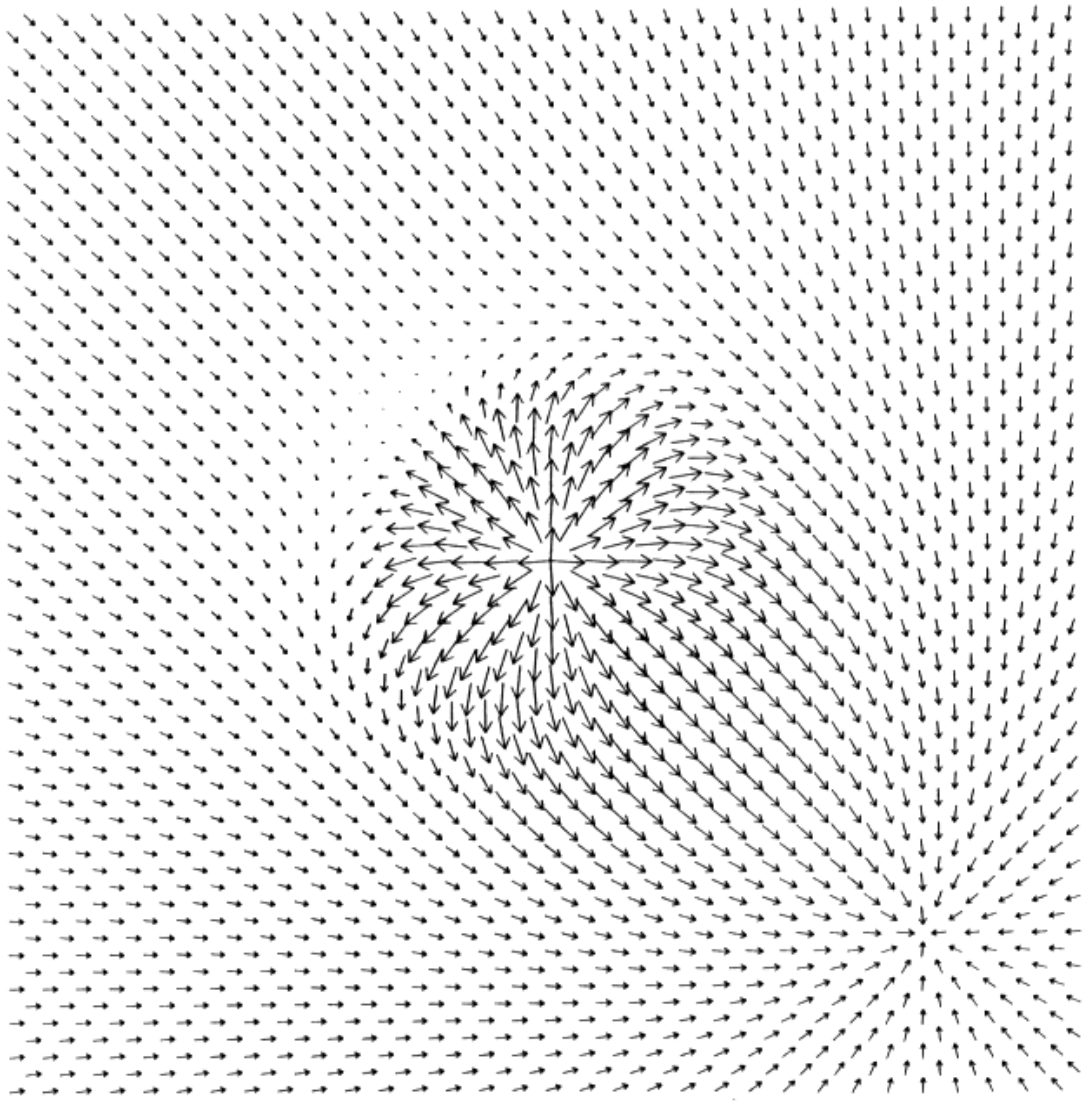
Potential Fields – Gradients



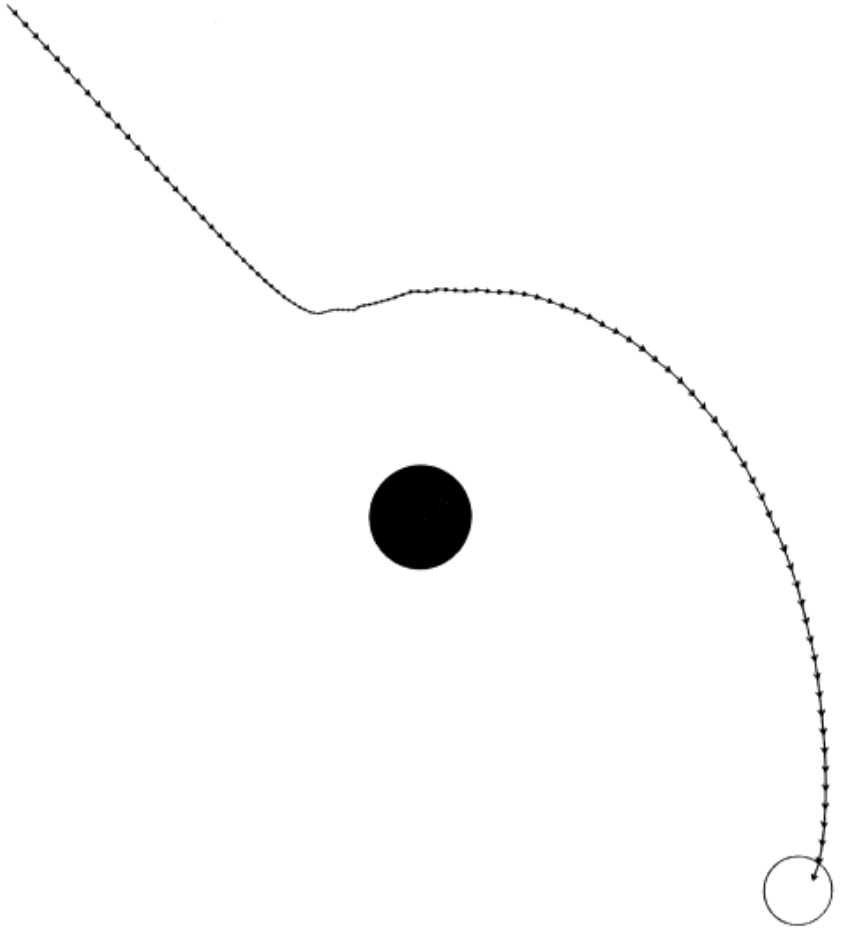
Potential Fields – Gradients



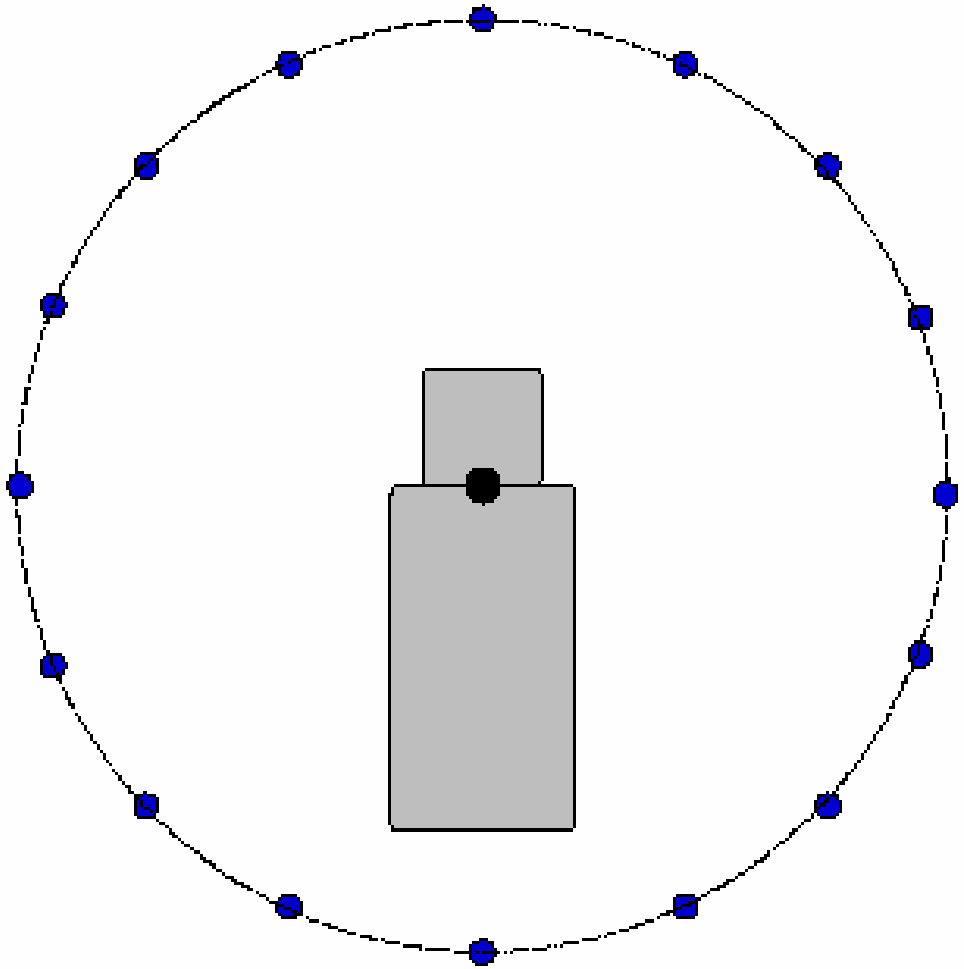
Potential Fields – Gradients



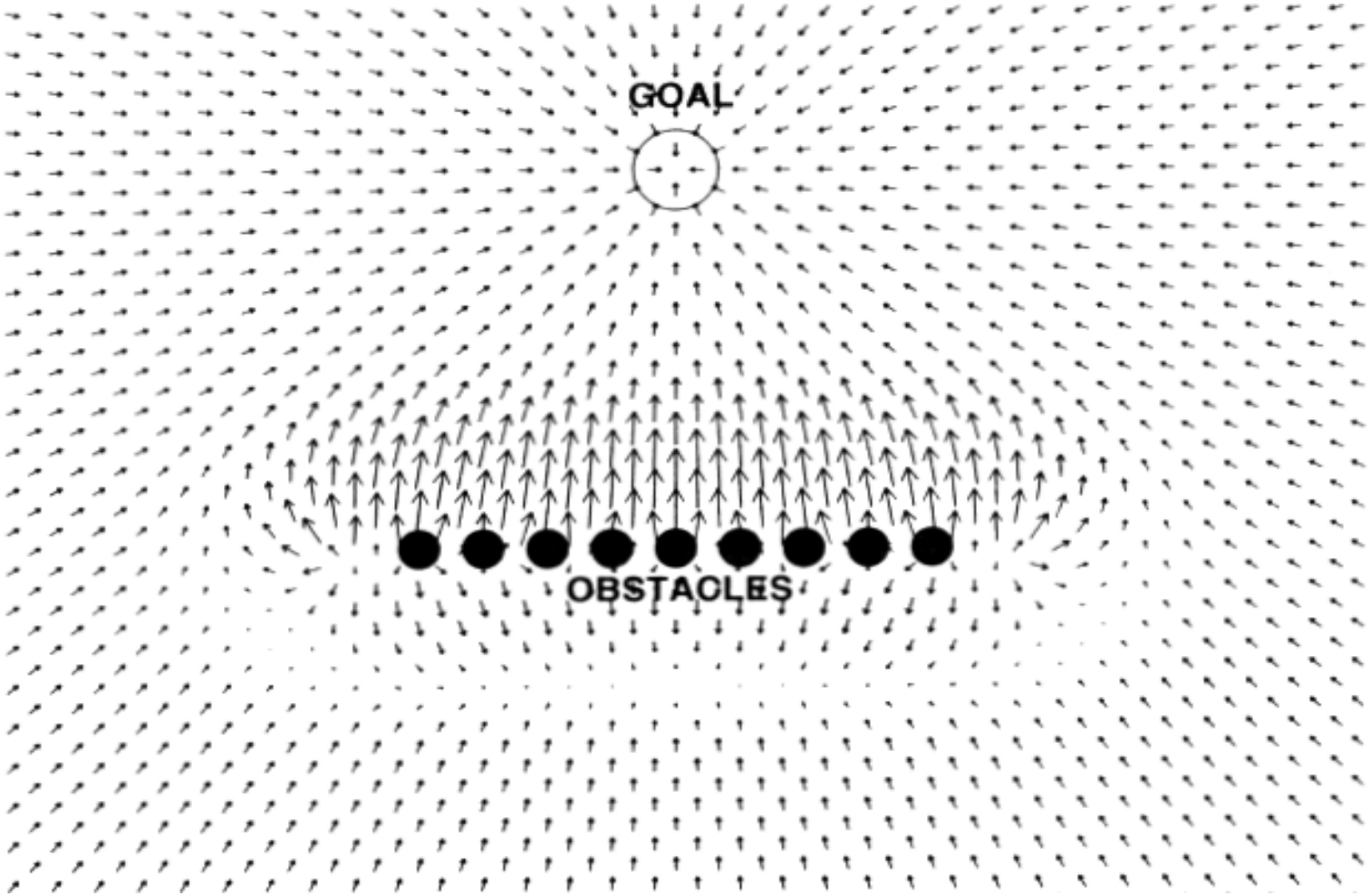
Potential Fields – Gradients



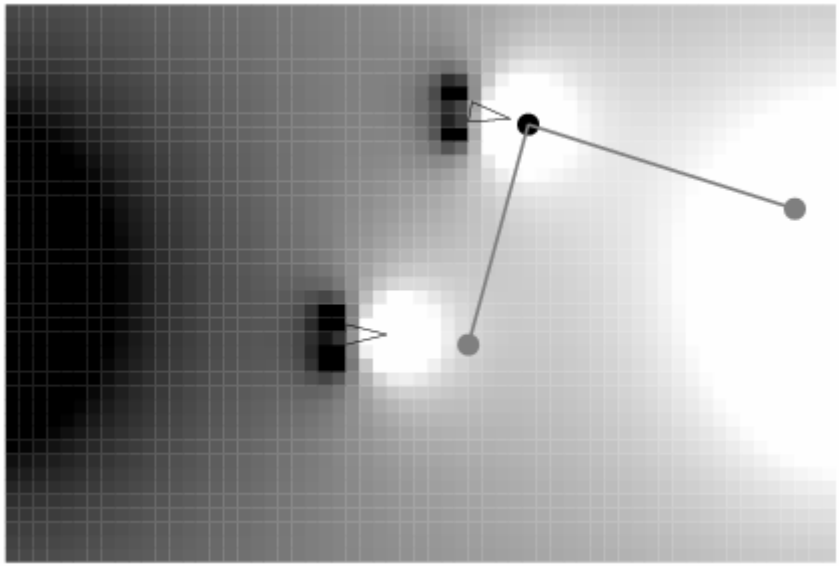
Potential Fields – Sampling

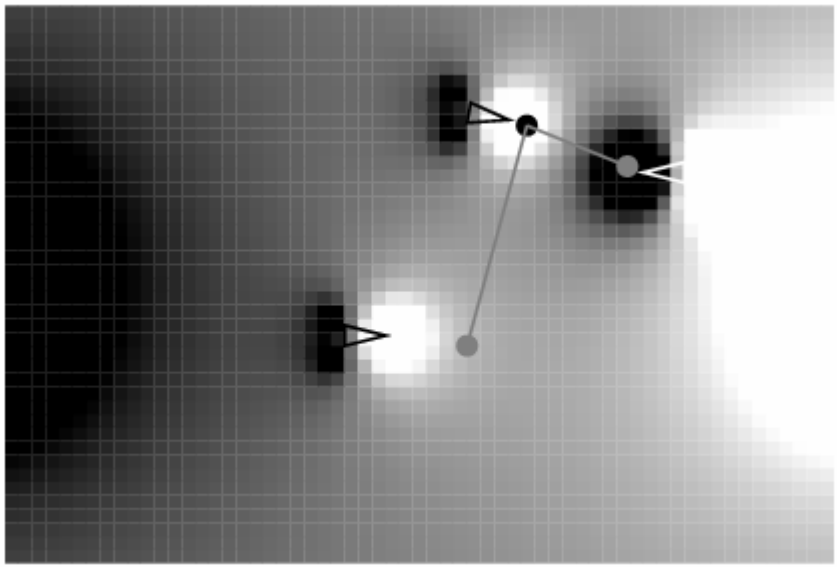


Potential Fields – Local Minima



Potential Fields – Action Planning

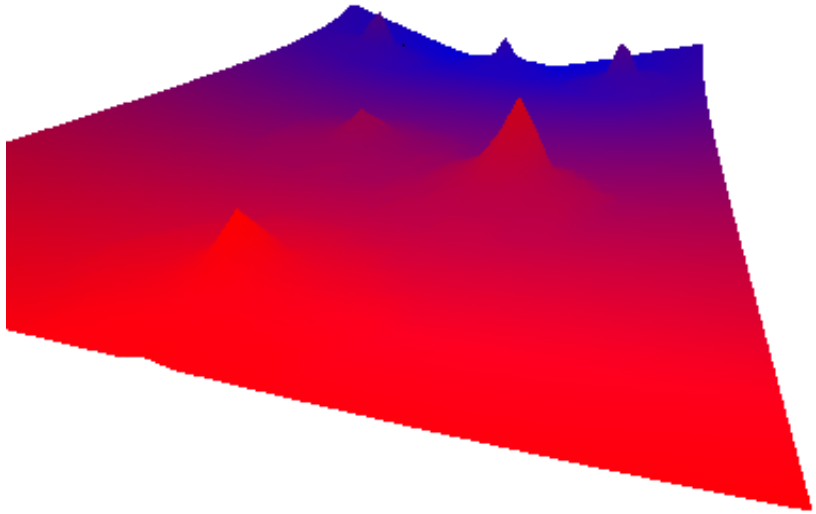


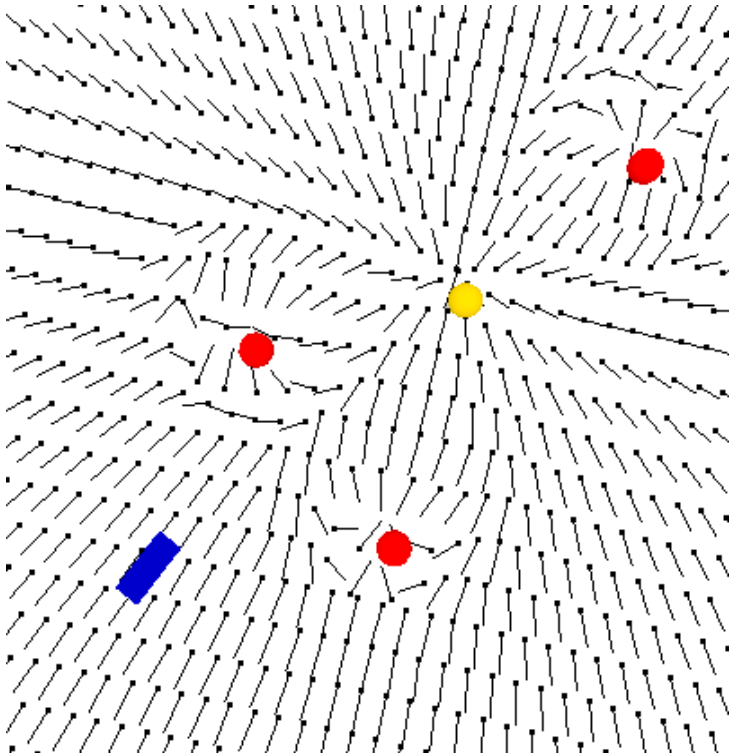


Potential Fields – Definition in XML

```
<object name="Opponent-Robot" type="repulsive">
        <asymptotic-function range="200"
                             at-zero="1000"
                             const-interval="1"/>
        <point-field/>
        <circle radius="90"/>
</object>

<motionfield name="go-to-ball">
       <return-const value="-1"/>
       <include name="ball"/>
       <include name="own-penalty-area"/>
       <include-group name="all-robots"/>
</motionfield>
```

Potential Fields – Visualization





Actuatorics – Mof

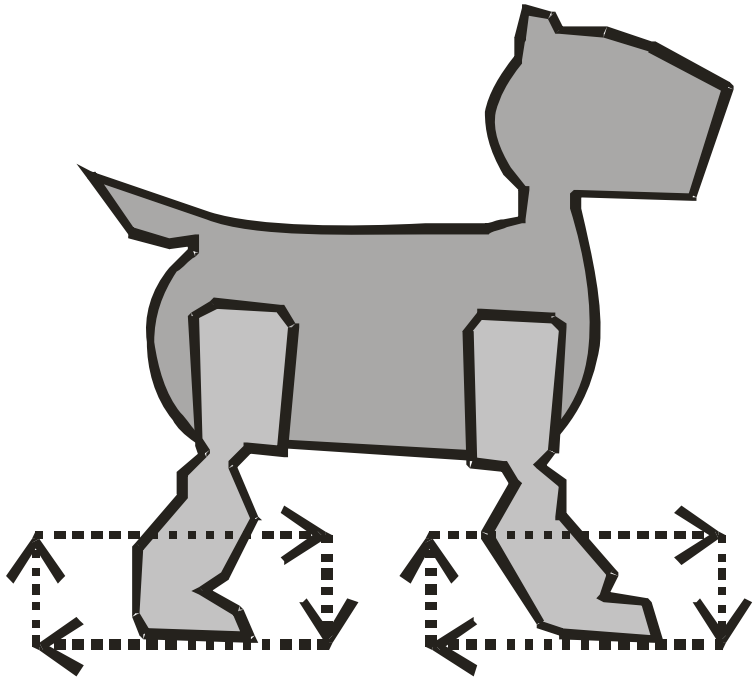
```
motion_id = unswBash
label start
//catch ball
0 0 0   ~ ~ ~   1034 300 1000  1034 300 1000   -500 2500 2600  -500 2500 2600   0 25
0 0 0   ~ ~ ~   1034 800 1000  1034 800 1000   -500 2500 2600  -500 2500 2600   0 40
label fromhold
-300 0 0   ~ ~ ~   1034 -700  500  1034 -700 500   -500 2500 2600  -500 2500 2600   0 40
//kick
0 0 0   ~ ~ ~   2550 0 2000  2550 0 2000   -500 2500 2600  -500 2500 2600   0 25
0 0 0   ~ ~ ~   2550 -400 2000  2550 -400 2000   -500 2500 2600  -500 2500 2600   1 15
0 0 0   ~ ~ ~   1034 -400 500  1034 -400 500   -500 2500 2600  -500 2500 2600   0 30
//stand
0 0 0  ~ ~ ~ -100 200 2100  -100 200 2100   -700 250 1500  -700 250 1500   0 25
0 0 0   ~ ~ ~   -115 262 1913  -70 256 1924  -579 222 1382 -547 188 1384  0 25
transition allMotions extern start
```

Parameters:

- 3 x Head
- 1 x Mouth
- 2 x Tail
- 3 per leg
- Interpolation
- Duration

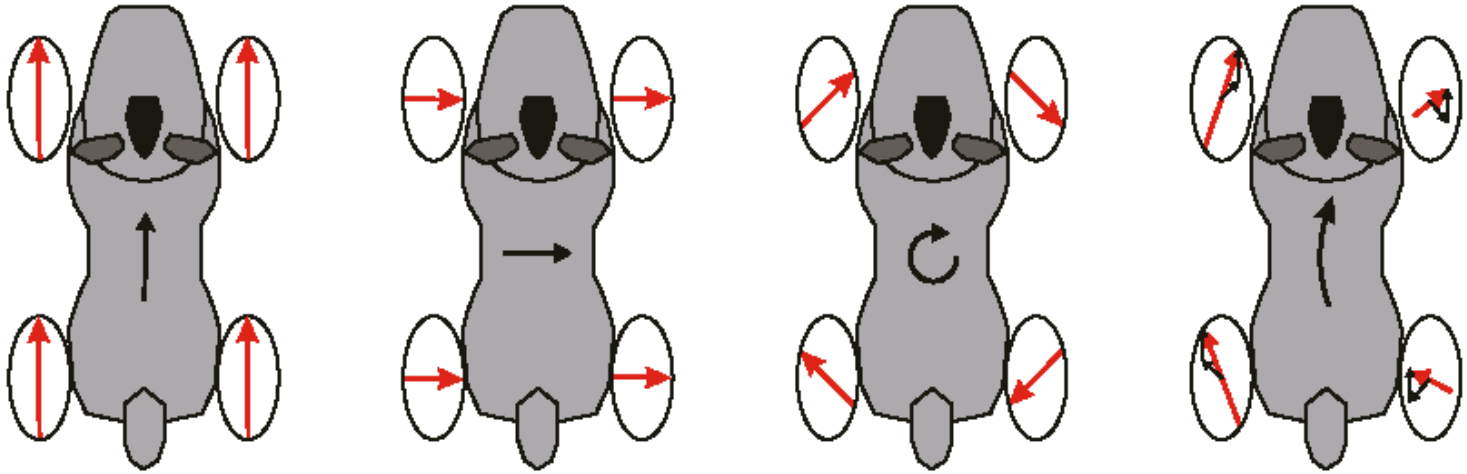
Actuatorics – Walking engine

- Robot´s maximum speed: ~23 cm/s
  - Requested speed defines the step size
  - Joint angles are generated every 8 ms
  - Foot motion: e.g. rectangular



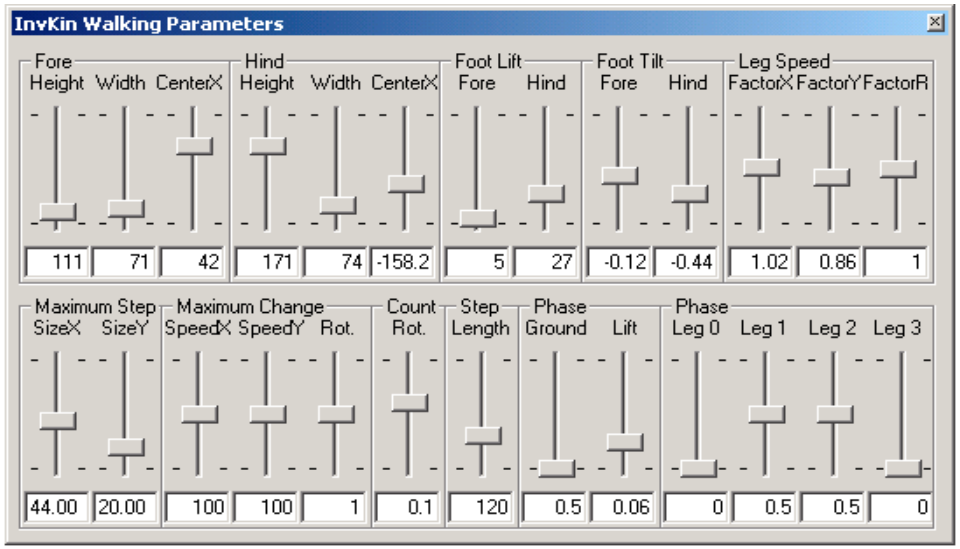
Actuatorics – Walking engine

- Direction of foot movement defines walk direction
- Trot walk
  - diagonal foot pairs move at the same time
  - Both pairs are shifted by a half step



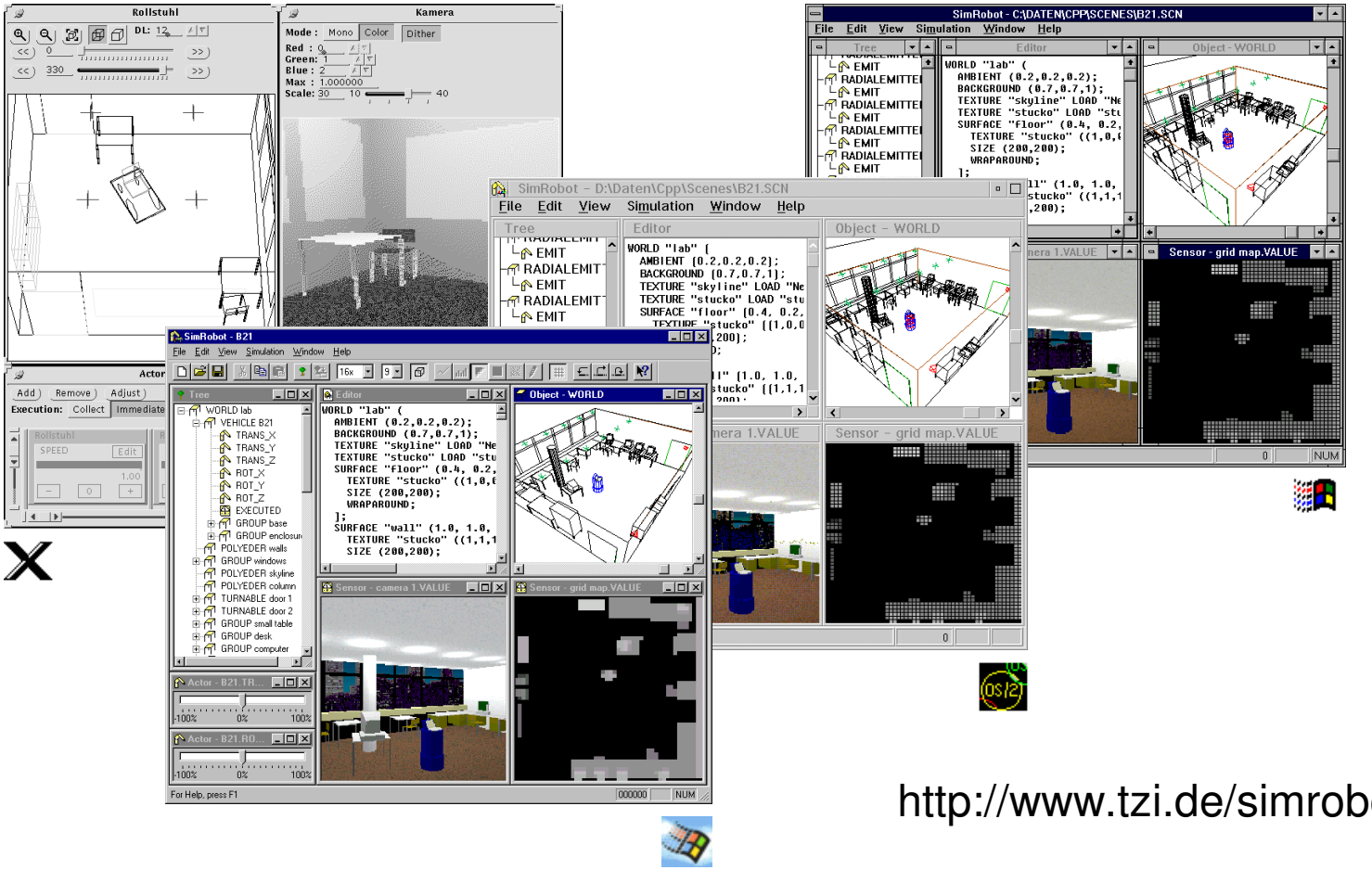
Actuatorics – Walking engine

- 30 parameters
  - max. step size
  - step height
  - step length
  - …



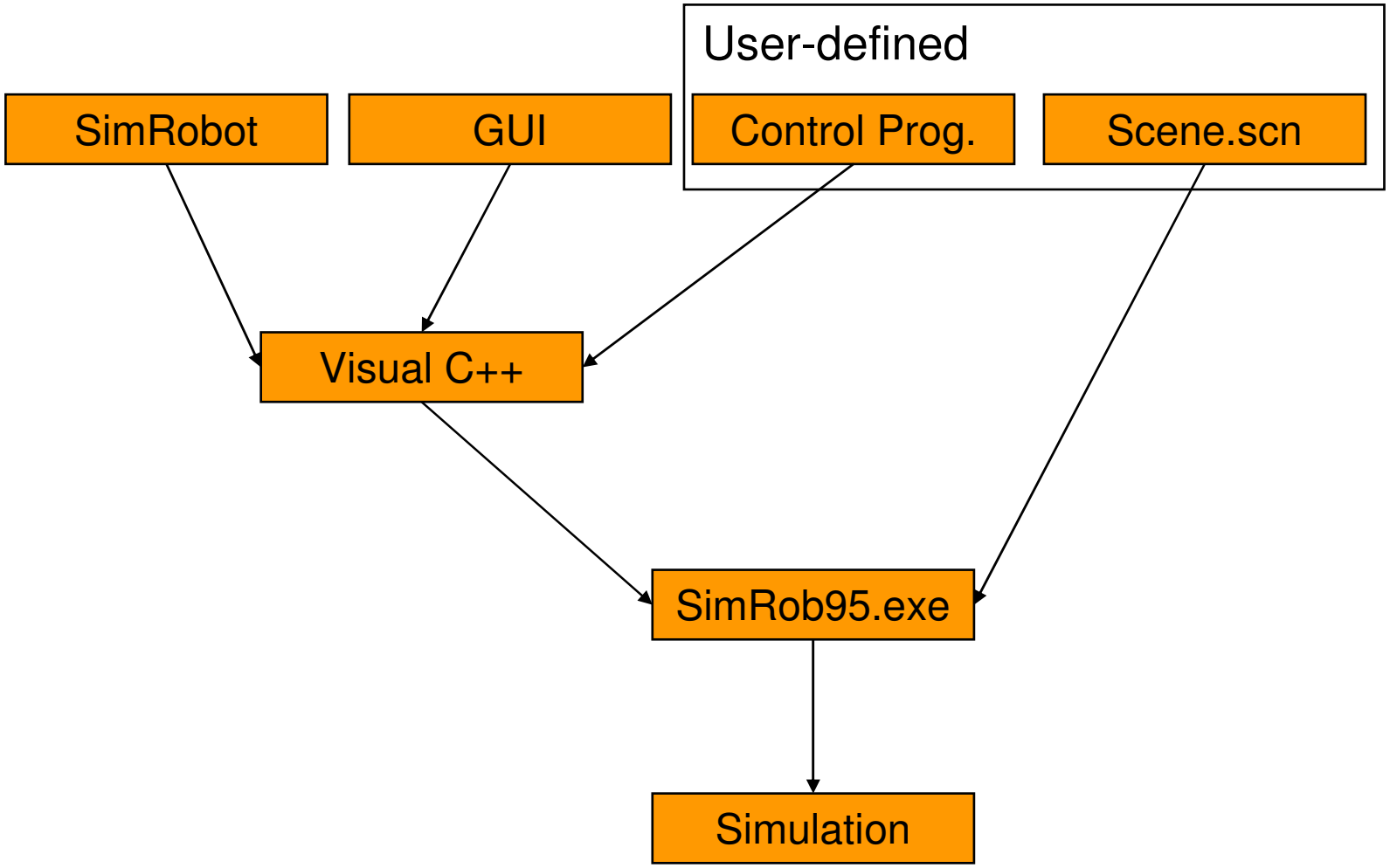
- Different parameter sets cause different walk types

Tools – SimRobot

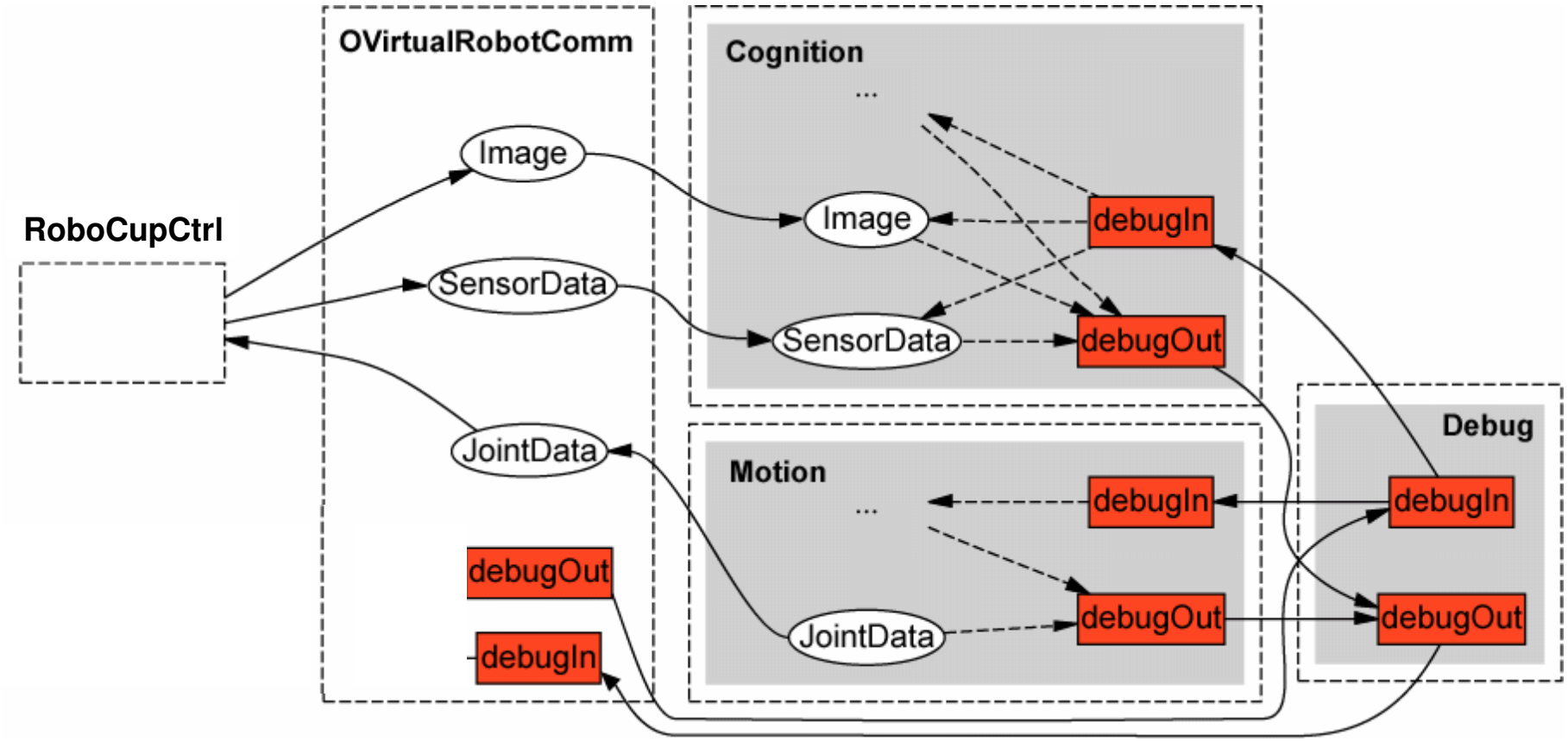


http://www.tzi.de/simrobot

Tools – Structure of SimRobot



Tools – SFLRL Simulation in SimRobot



Tools – SimRobot Demo