Robóts autónomos y la RoboCup

José María Cañas Plaza

jmplaza@gsyc.escet.urjc.es

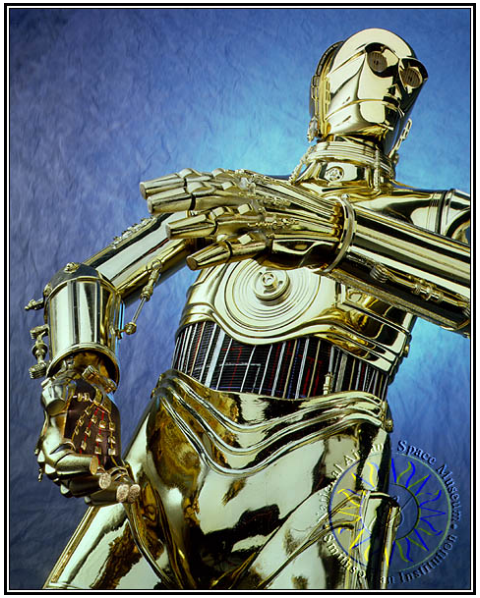
Curso de Verano, U.Santiago de Compostela
julio 2004

Índice del curso

- Introducción
- RoboCup
- Simulación
- EyeBot
- Aibo

Introducción

Ciencia Ficción



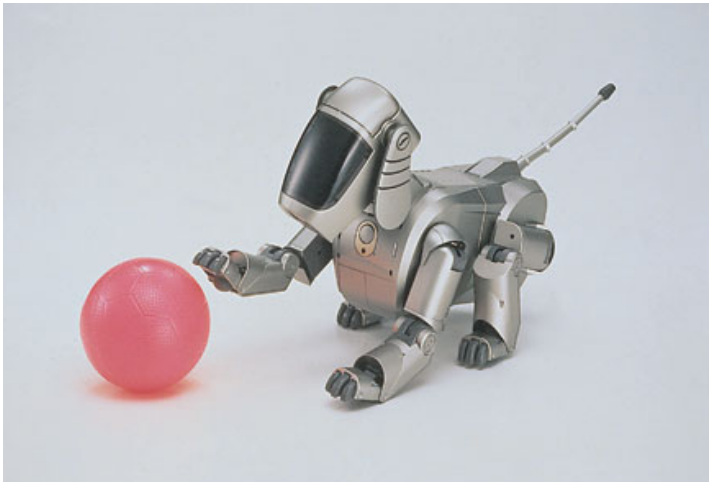
Robótica real

Robótica

Conseguir que los robots hagan cosas

- Componente tecnológica
- Utilidad industrial
- Investigación en autonomía
- Ingenierías informática, industriales, telecomunicaciones
- Control automático

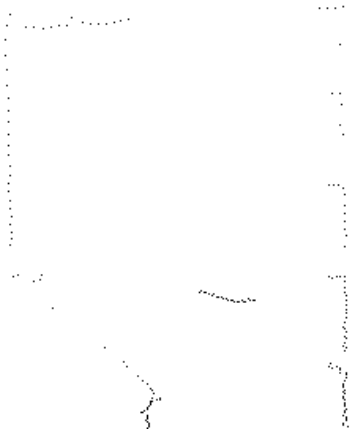
Robótica en los hogares



¿Qué es un robot?

- Sensores
- Actuadores
- Computador
- Programación

Sensores

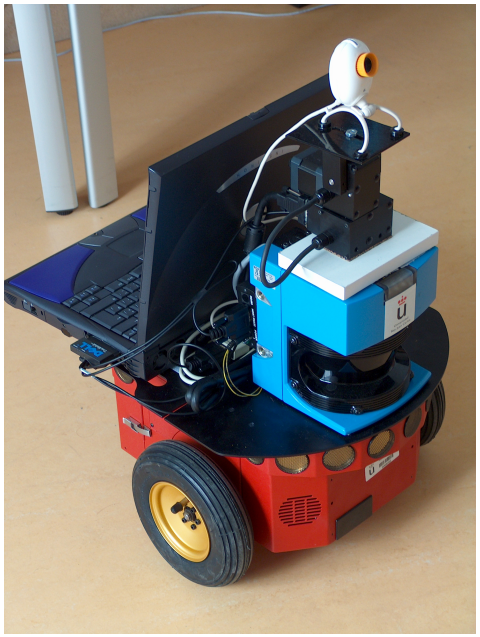


- activos/pasivos
- externos/internos
- luz
- contacto
- odómetros
- sónares, laser, infrarrojos
- cámaras

Actuadores

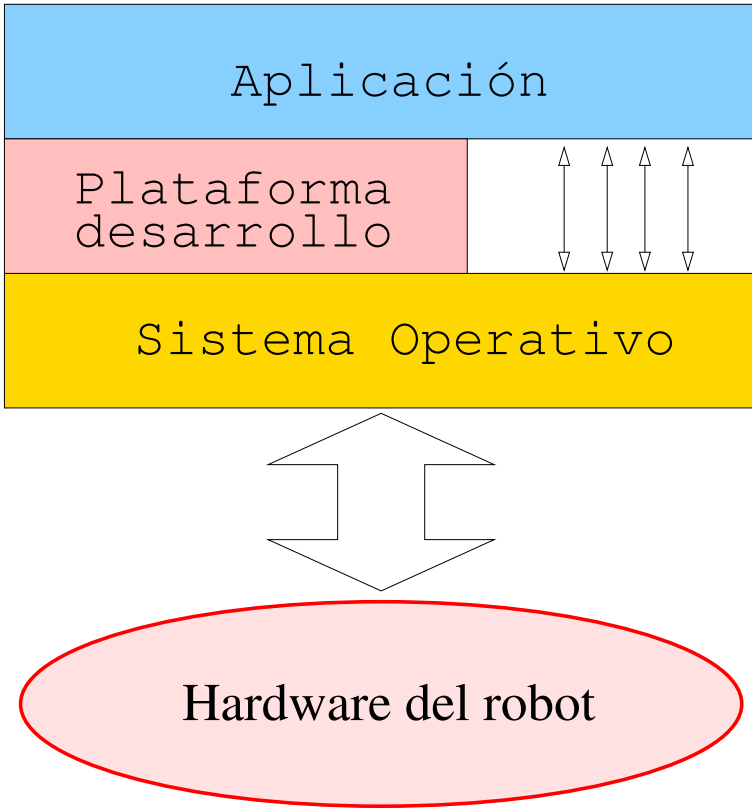
- eléctricos
- pneumáticos
- hidráulicos
- servos
- motores
- válvulas

Procesadores



- microcontroladores
- PC, sobremesa o portátiles

Programación de robots móviles



- hardware, realidad física
- sistema operativo
- plataforma desarrollo
- aplicaciones
- lenguajes
- no hay estándares

Grupo de robótica URJC

http://gsyc.escet.urjc.es/robotica

curso	gente	robots
1999-2000	1 doct	+20 Lego, +2 EyeBots
2000-2001	1 doct 1 ay	+4 EyeBots, +10 Lego
2001-2002	1 doct 1 ay	+1 Pioneer
2002-2003	1 doct 2 ay	+1 Pioneer + 1 laser
2003-2004	2 doct 2 ay 1 vis 3 bec	+3 Aibo + 1 laser


Carnegie Mellon

RoboCup

Índice

- ¿Qué es?
- Categorías
- Historia
- Banco de pruebas

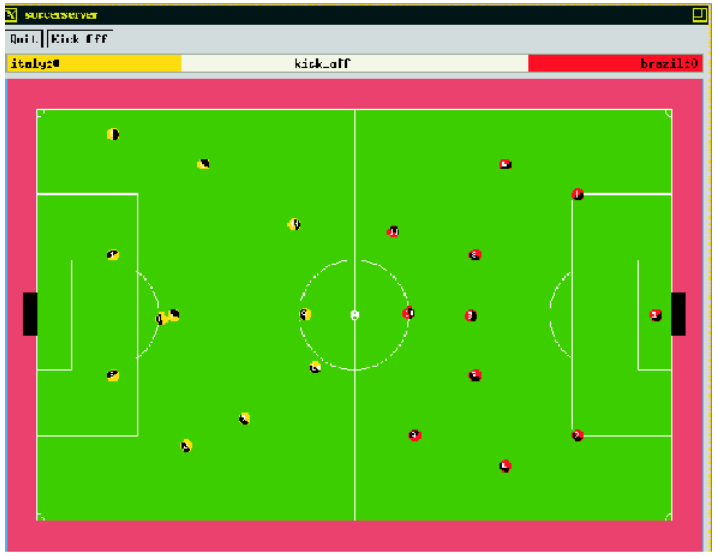
Introducción a la RoboCup

http://www.robocup.org

- Impulsar la investigación en IA
- Marco común: fútbol robótico
- RoboCup Federation: congresos, torneos, campus, etc.
- Conjunto de actividades para la promoción de la investigación en sistemas inteligentes.
- fútbol: varias categorías
- no-solo-fútbol: rescue, junior

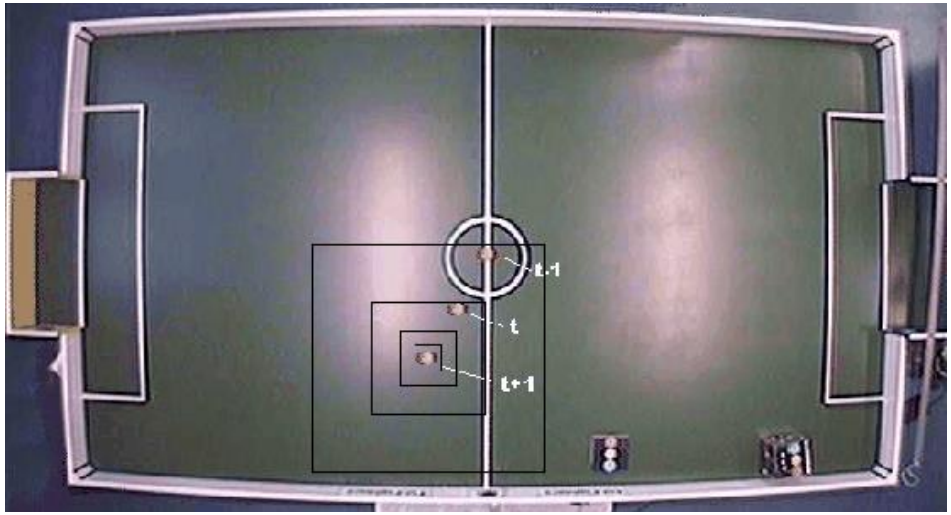
Categorías

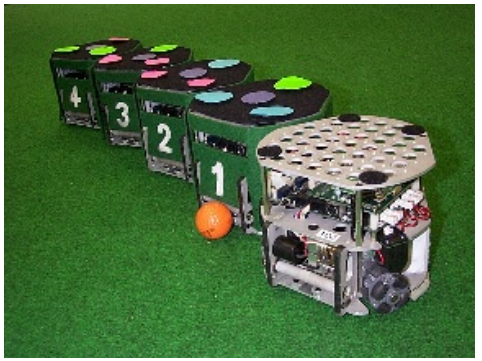
Categoría simulada



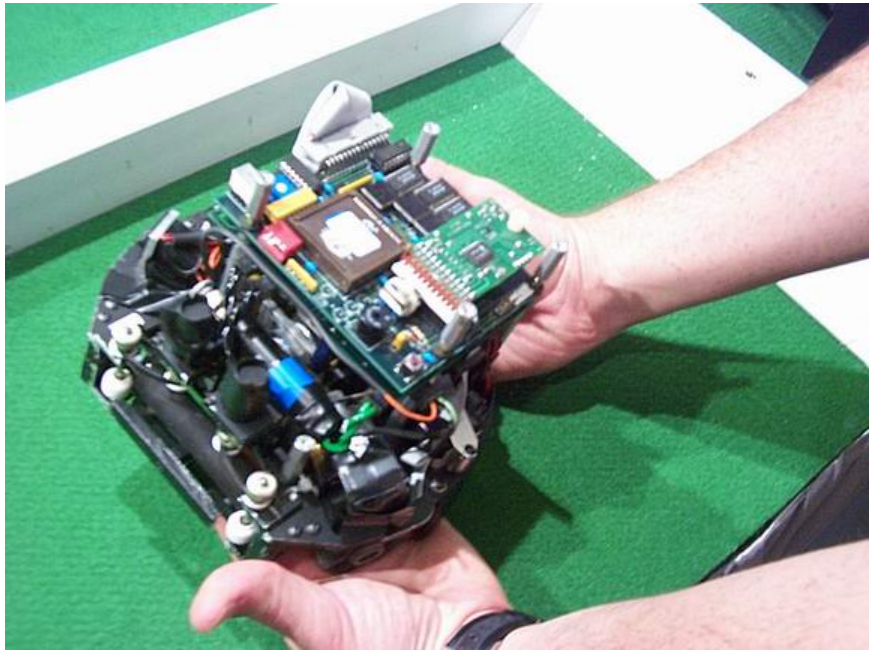
- no hay problemas de hardware
- balizas para localización
- comunicación interjugador fácil
- cooperación
- todos mismo "hardware"
- subjetivo

Categoría pequeña, F-180





- F-180
- visión cenital
- posición y orientación
- procesamiento remoto

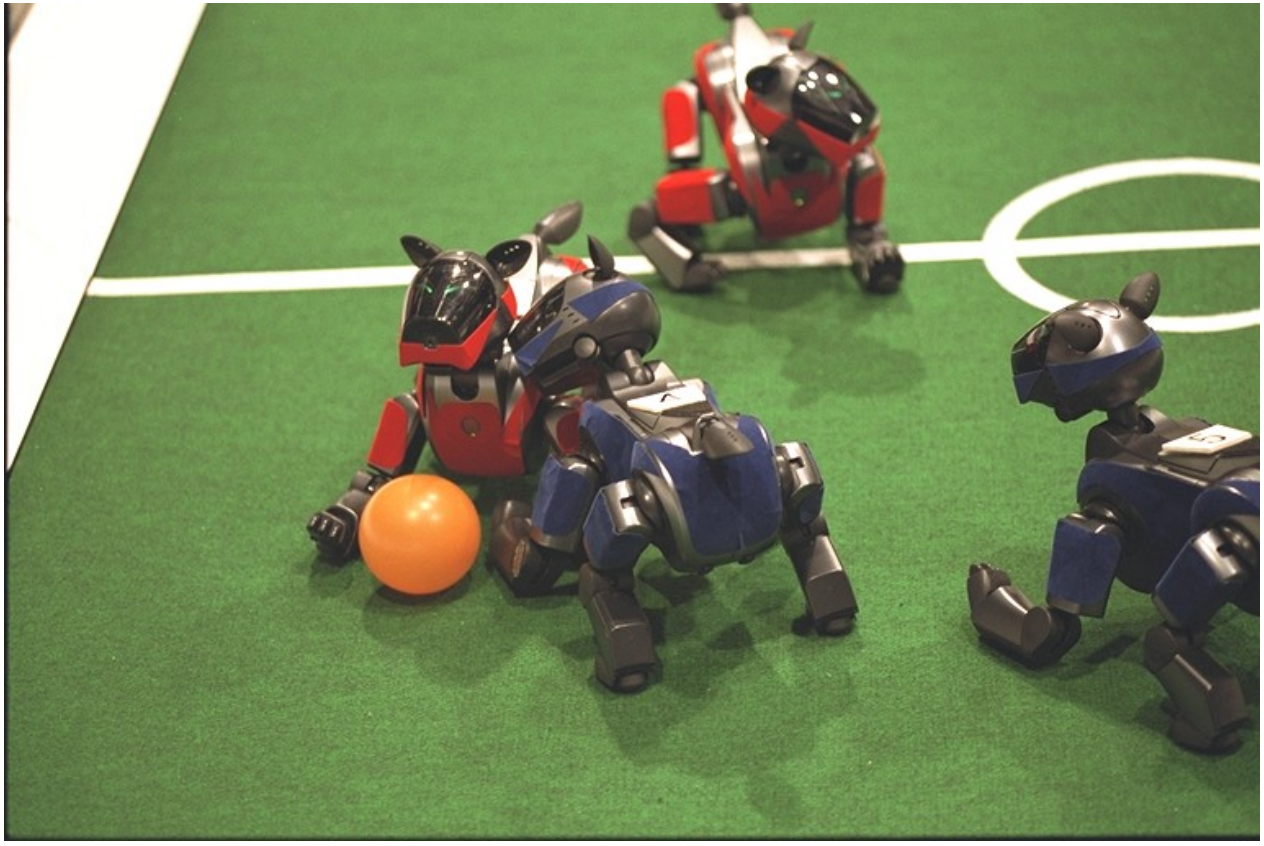


- ordenador personal
- comunicación interjugador sencilla
- análisis vivaz de imágenes es clave
- rapidez movimientos
- construcción mecánica
- con/sin chutador

Categoría mediana, F-2000

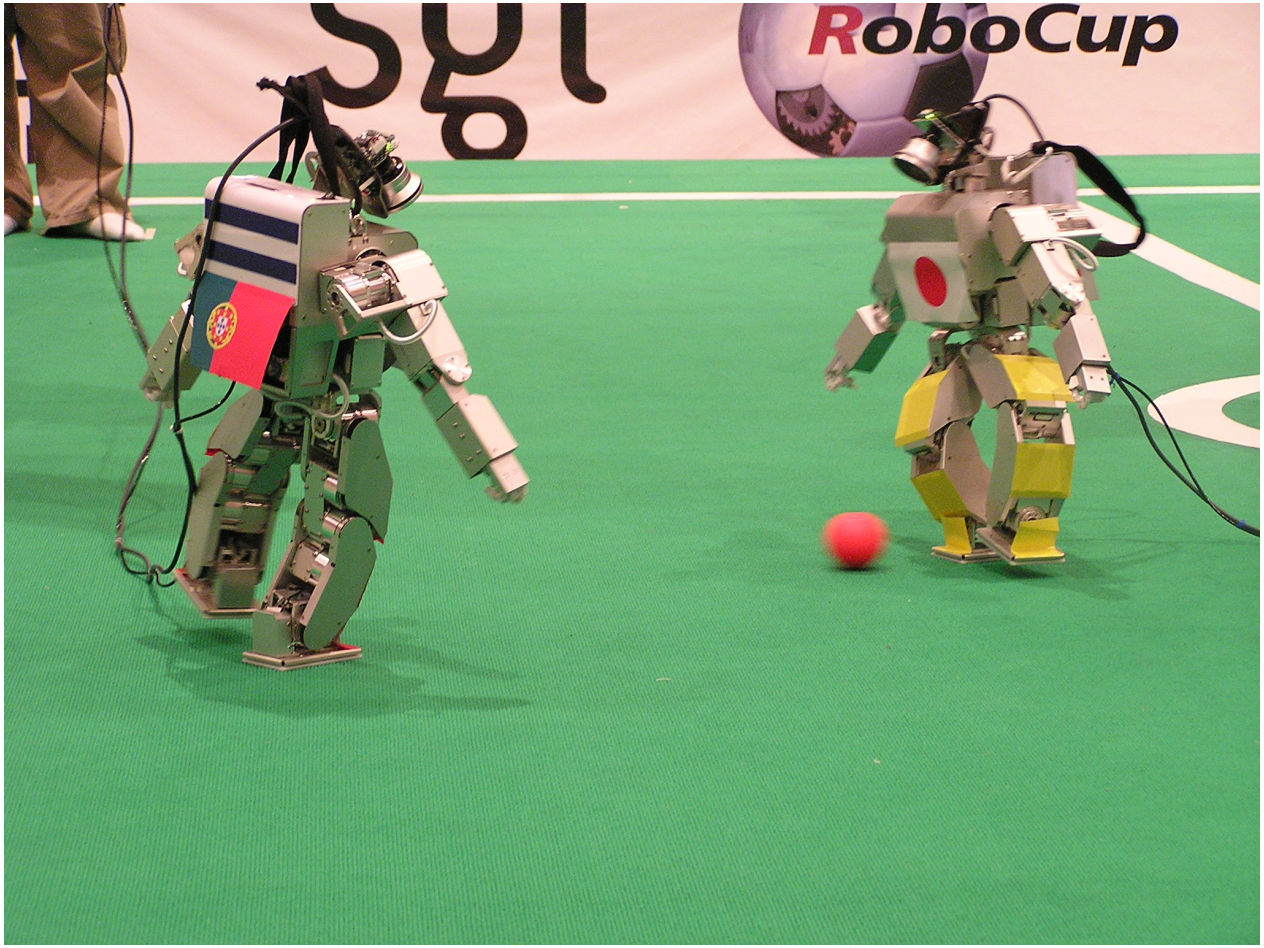
- visión local, omnidireccional
- comunicación interjugador difícil
- rapidez movimientos
- construcción mecánica
- con/sin chutador

Categoría cuatro patas



- visión local, atención
- balizas para localización
- comunicación interjugador difícil
- misma plataforma hardware
- codos

Bípedos



Historia y evolución

- 1997 Nagoya, Japón
- 1998 París, Francia
- 1999 Estocolmo, Suecia
- 2000 Melbourne, Australia
- 2001 Seattle, USA
- 2002 Fukuoka y Busan, Japón y Korea
- 2003 Padua, Italia
- 2004 Lisboa, Portugal

- cada año van complicando el problema
- bordes, cuello, bípedos...
- 2050, competir contra el campeón humano
- se notan los progresos año a año

Banco de pruebas

- escenario real, dinámico, competitivo
- control basado en visión
- localización
- selección de acción, arquitectura
- atención
- coordinación y cooperación: pase, equipos
- IA distribuida, agentes

Otras competiciones de robots

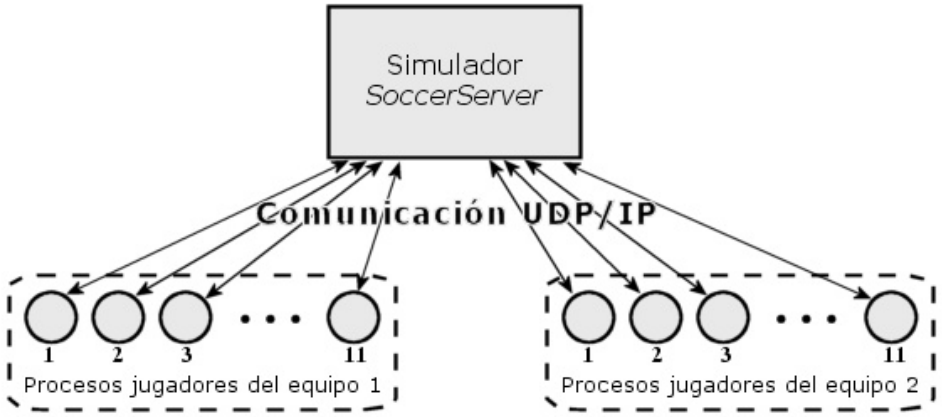
- Mirosot: www.fira.net
- Hispabot: www.depeca.uah.es/alcabot/hispabot2004
- Robocampeones: www.robocampeones.com

Simulación

Índice

- Soccerserver
- ABC^2
- Equipo con lógica borrosa
- Equipo con JDE
- Localización con visión

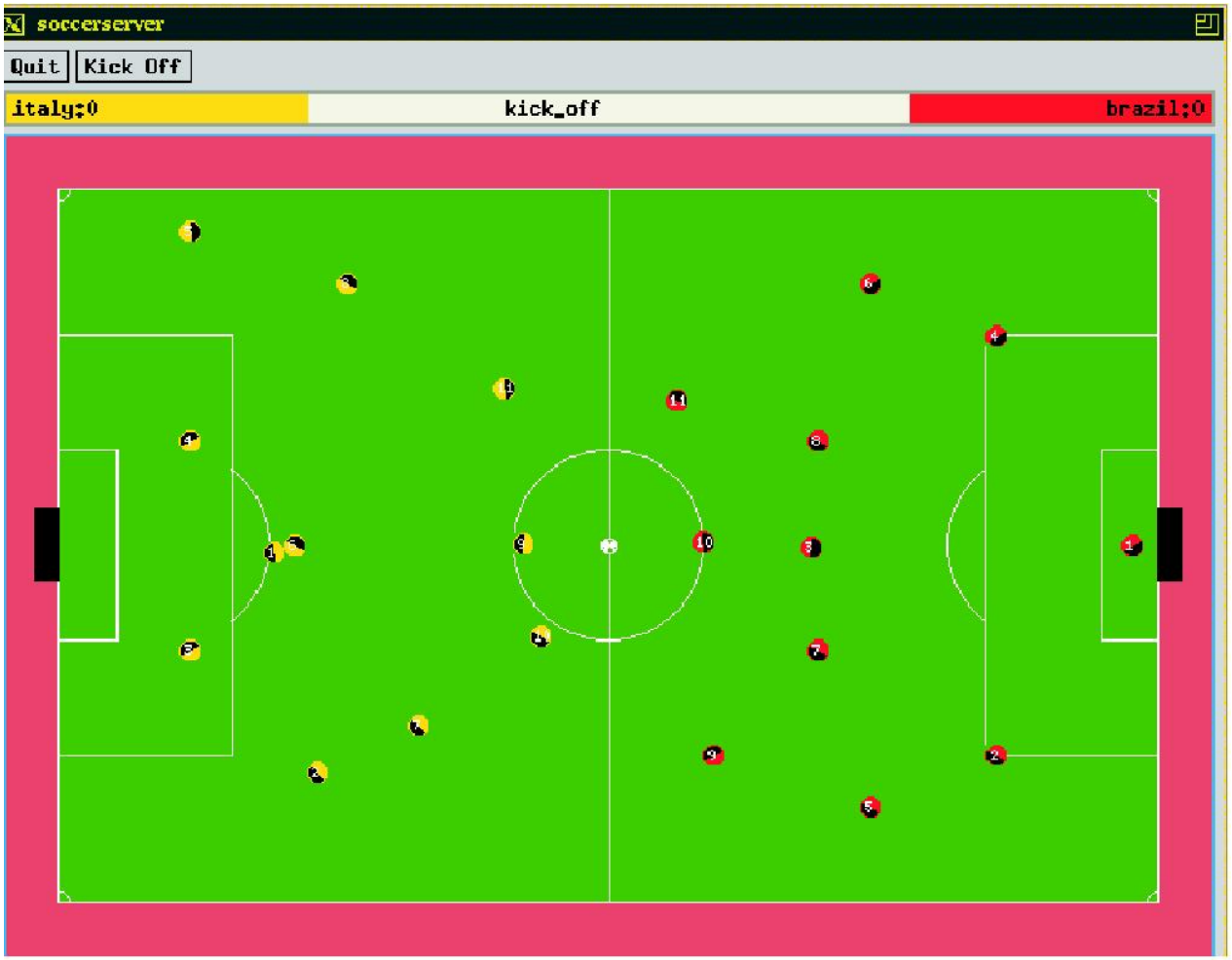
Soccerserver



- Cada jugador es un proceso independiente ejecutando:
  - Procesamiento de la información sensorial
  - Evaluación de su sistema de control
  - Envío de las acciones al simulador: movimiento y comunicación
- Cada jugador puede tener su propia memoria: introspectiva y externa

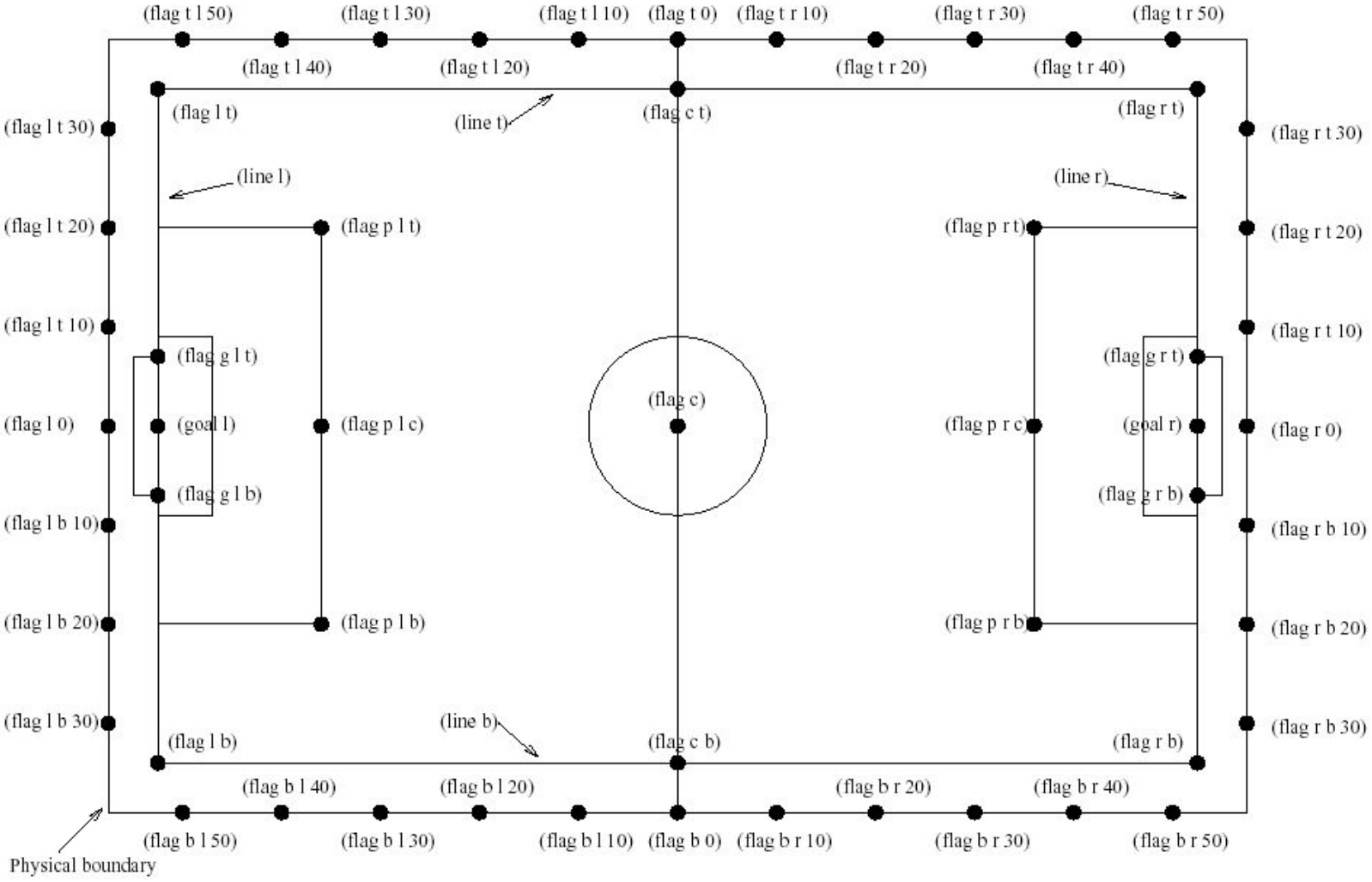
- Cada jugador recibe de forma indiviualizada su percepción subjetiva (en forma de información visual y sonora pre-procesada)
- Cada jugador debe envíar el resultado de su proceso de razonamiento en forma de órdenes de movimiento
- Un árbitro humano garantiza el correcto desarrollo del juego, además de la existencia de uno automático (fueras de juego, de banda,etc.).
- El simulador de la RoboCup está especialmente pensado para la simulación de agentes. El modelo de programación es absolutamente flexible

El partido



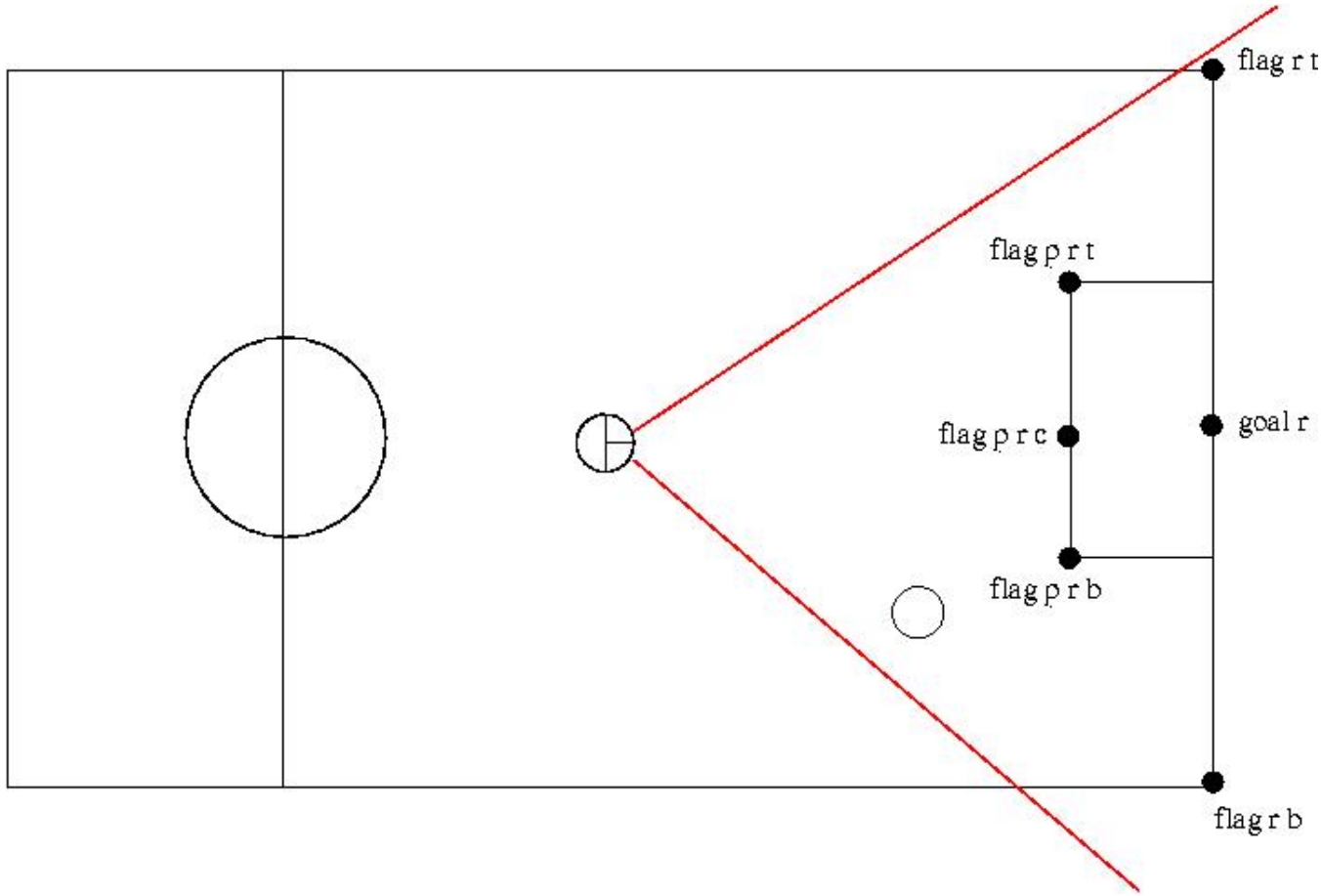
- El simulador es un proceso independiente, al igual que cada uno de los 22 jugadores
- Adicionalmente existen procesos auxilares que permiten realizar diversas tareas. Por ejemplo, el **player** permite visualizar los partidos.
- Existen diferentes simuladores y visualizadores en 2 y 3 dimensiones
- También existen herramientas para "grabar" los partidos y reproducirlos para poder analizarlos.

Objetos del mundo



- La información visual recibida por cada jugador se refiere a los jugadores, la pelota y una serie de "marcas" que definen el campo, que entren dentro de su campo visión según su posición y orientación.
- Las marcas permiten a los jugadores *localizarse* en el campo, puesto que esa información no se suministra directamente a los jugadores.
- La interpretación de las marcas del campo se hace de la siguiente forma:
  - `flag r b`: bandera derecha (`r`ight), abajo (`b`ottom).
  - `flag g r t`: bandera de la línea de gol derecha (`r`ight) arriba (`t`op).

Cono de visión



- El cono admite una pequeña configuración, permitiendo elegir entre un cono más estrecho y de mayor precisión, o un cono de visión más ancho pero con mayor precisión.
- En las última versiones además se ha añadido la funcionalidad de "cuello" a los agentes, lo que permite que los jugadores puedan desplazarse en una dirección y "mirar" en otra.
- La información visual tiene ruido, que se incrementa con la distancia (a más distancia al objeto, más ruido).

Ejemplo de información

```
recv 2035: (see 0 ((goal r) 73 -7) ((flag r t) 84.8 -31)
    ((flag r b) 76.7 18) ((flag p r t) 63.4 -28) ((flag p r c)
     56.8 -10) ((flag p r b) 56.8 10) ((ball) 22.2 -26 0 0)
     ((line r) 72.2 -90))
```

- Tiempo transcurrido del partido (2035)
- Tipo de información recibida (`see`)
- Lista de elementos percibidos, al estilo LISP.(la portería derecha, (`goal r`), una serie de banderas, la bola (`ball`), etc.)
- Para los elementos estáticos se indica la distancia y dirección relativa. Para los móviles (la bola y los jugadores) se indica además la velocidad relativa y su variación, para dar una idea aproximada de su movimiento

Acciones

- Los clientes pueden realizar una serie de acciones básicas
- Se implementan como mensajes a servidor
- *turn*, *turn_neck*, *dash*, *kick*, *move*, *catch*, *say* y *change_view*
- En un ciclo de simulador, sólo se puede realizar un movimiento
- El jugador deberá responder con un comando de sintaxis similar en el que se indica cuál es la acción que desea ejecutar (p.e. girar `turn`)

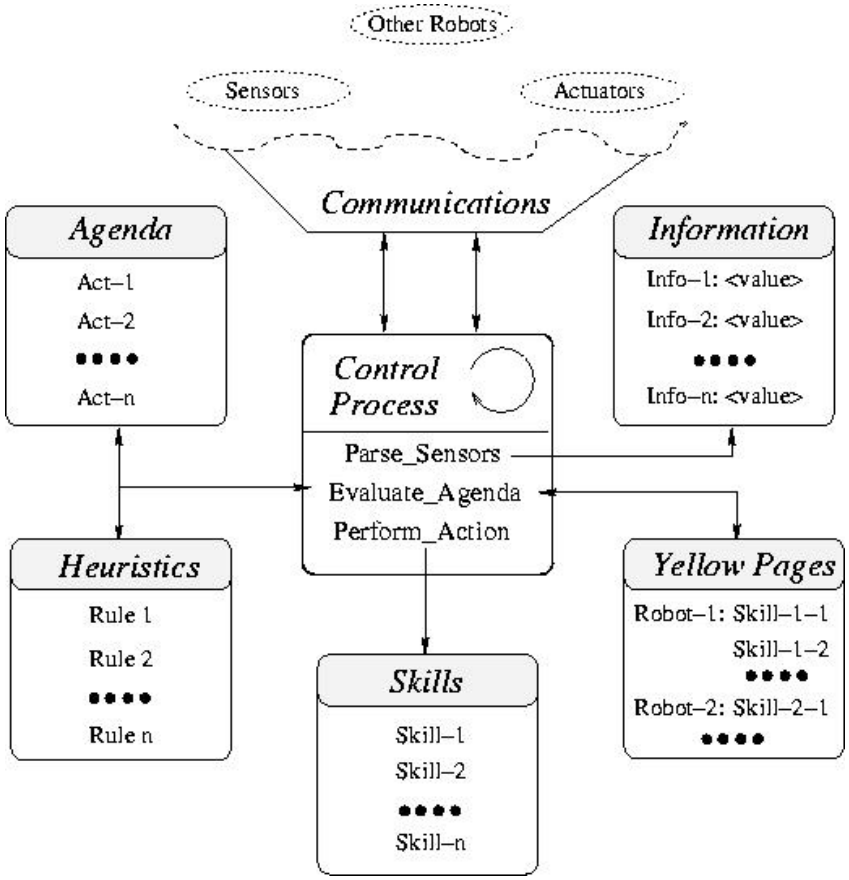
Un ejemplo de equipo: ABC^2

Un agente (A) se representa por la tupla: $A =< N, S, Y, L, H >$

$\texttt{robot\_1} = < \texttt{robot}_1,$
$\{\texttt{Go\_Ball}, \texttt{Look\_for\_Ball}, \ldots, skill_{1n}\},$
$\{\texttt{robot}_2\text{: Pass}, \texttt{Go\_Ball}, \ldots skill_{2m}\},$
$\{\ Ball\text{: (32,50)}, \ldots Concept_k\text{: } (value_{k_1}, \ldots valune_{k_i}),$
$\quad Attacking, \ldots info_j\ \}$
$\{\ (Ball, Goal_{own}, \ldots, Concept_k)^+,$
$\quad (Attacking, Defending \ldots info_l)^+\ \},$
heuristic-rules $>$

- La arquitectura ABC^2 (Matellán98) que se basa en el uso de una pizarra para elegir la acción a ejecutar entre un repertorio de habilidades (S)
- Cada jugador tiene su propia pizarra
- Los jugadores pueden intercambiar mensajes: do,request,inform
- Las heurísticas (H) que eligen entre las habilidades tienen en cuenta la información local del agente (L).
- Heurísticas borrosas resuelven la selección de acción
- Descomposición de tareas en subtareas

La arquitectura de los jugadores



Otro ejemplo de equipo: URJC-borroso

- El comportamiento de los jugadores se controla mediante unas reglas difusas que deciden que comportamiento ejecutar de entre un repertorio: $Mirar_Bola$, etc.
- Evaluación de un sistema borroso: comportamiento a ejecutar.
- Cada jugador tiene su propia memoria: introspectiva y externa
- Combinación heurística de acciones básicas (*dash*, *turn*..)
- *Mirar_Bola*, *Buscar_Meta*, *Ir_Bola*, *Chutar_Gol*, *Ir_Pos*, *Pasar*, *Salir*, *Despejar*, *Buscar_Pase*, *Avanzar_Bola*, *Deambular*

Selección de acción con reglas borrosas

- Cada jugador tiene un repertorio de acciones básicas diferentes
- El conjunto de informaciones que le lleva a decidir cuál activar es sin embargo el mismo: *DistBola, DistMeta,* , etc.
- Las reglas de selección de selección además de estas informaciones deben tener en cuenta la situación del juego: (falta, saque de banda, de esquina, de portería, etc.)
- El conjunto de reglas y comportamientos determina el "rol" del jugador (delantero, defensa, etc.)

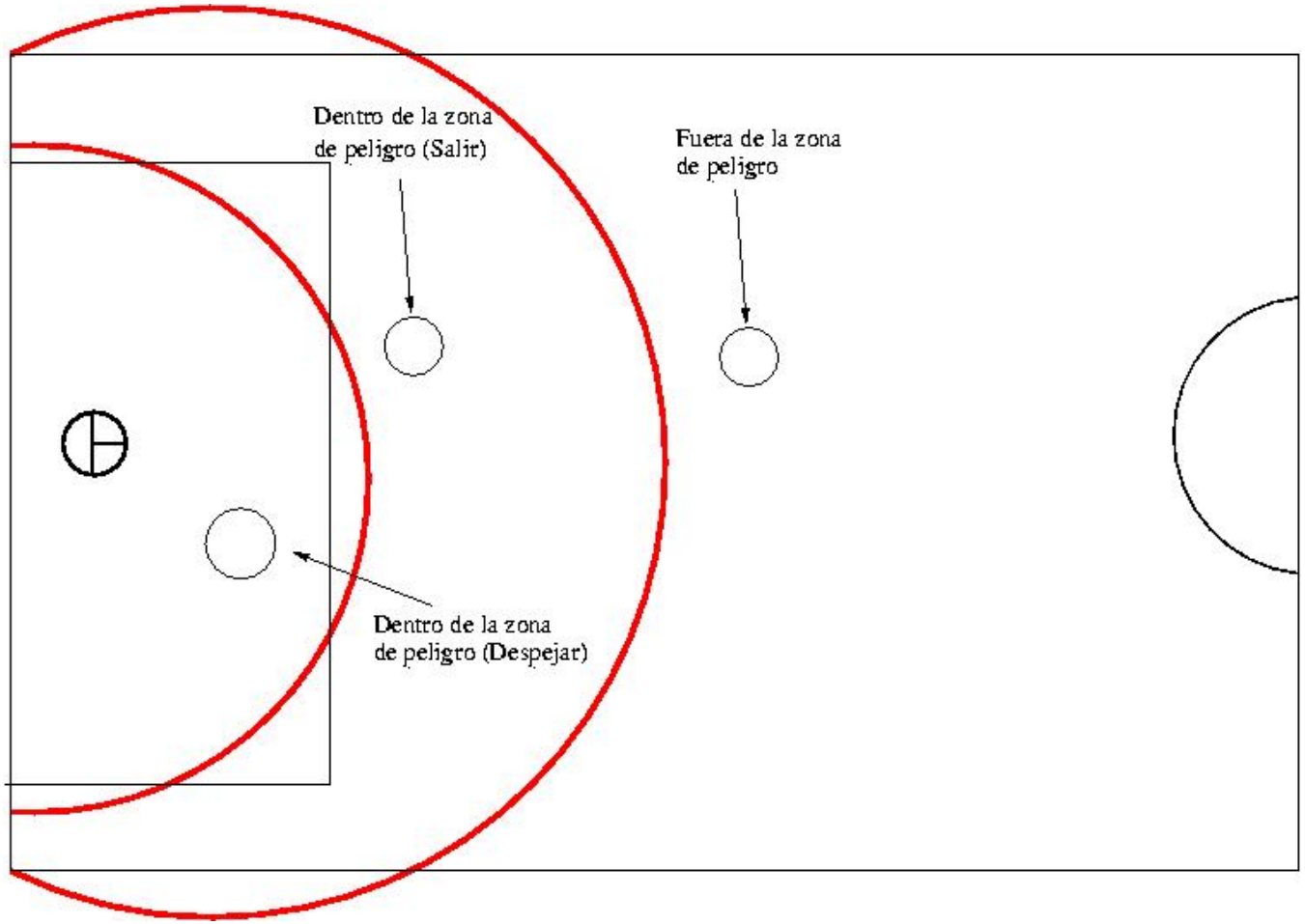
Variables de entrada

- Son iguales para todos los jugadores
- *DistBola*, *DistMeta*, contrario más cercano (*DistPosJuego*), ...
- Situación del juego (falta, saque de banda, ...)

Variables de salida

- Activación de comportamientos
- No activan el mismo subconjunto de comportamientos

Portero


Dentro de la zona
de peligro (Salir)
Fuera de la zona
de peligro
Dentro de la zona
de peligro (Despejar)

- En el caso del portero, p e, los comportamientos se activan en función de la posición de la bola respecto del portero.
- De forma similar se pueden asignar "zonas" de cobertura a los defensas o los delanteros, haciendo que los jugadores sean *sensibles* a la zona del campo en la que se encuentran.
- El uso de la información relativa a otros jugadores de su mismo esquipo (p.e. $DistComp =$ distancia al compañero más próximo) sirve para implementar los comportamientos de grupo.
- También se contempla el intercambio de roles: un defensa que pueda convertirse en delantero

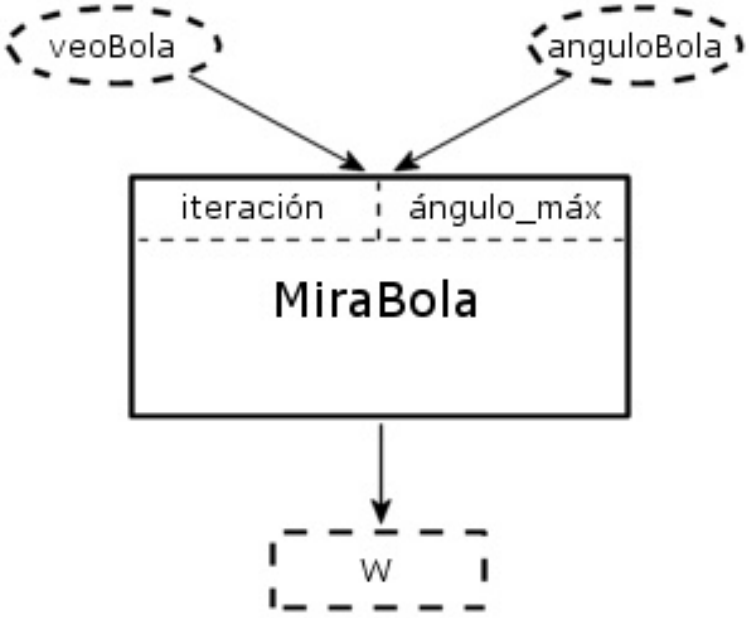
Otro ejemplo: URJC-esquemas

- Comportamiento como una combinación de percepción-actuación
- Ambos problemas se dividen en pequeñas partes con identidad propia, a las que se denomina *esquemas*
- Un esquema es un flujo de ejecución iterativo que tiene un objetivo concreto.
- Es modulable y puede ser activado y desactivado en cualquier momento.
- Los esquemas tienen asociado un estado:

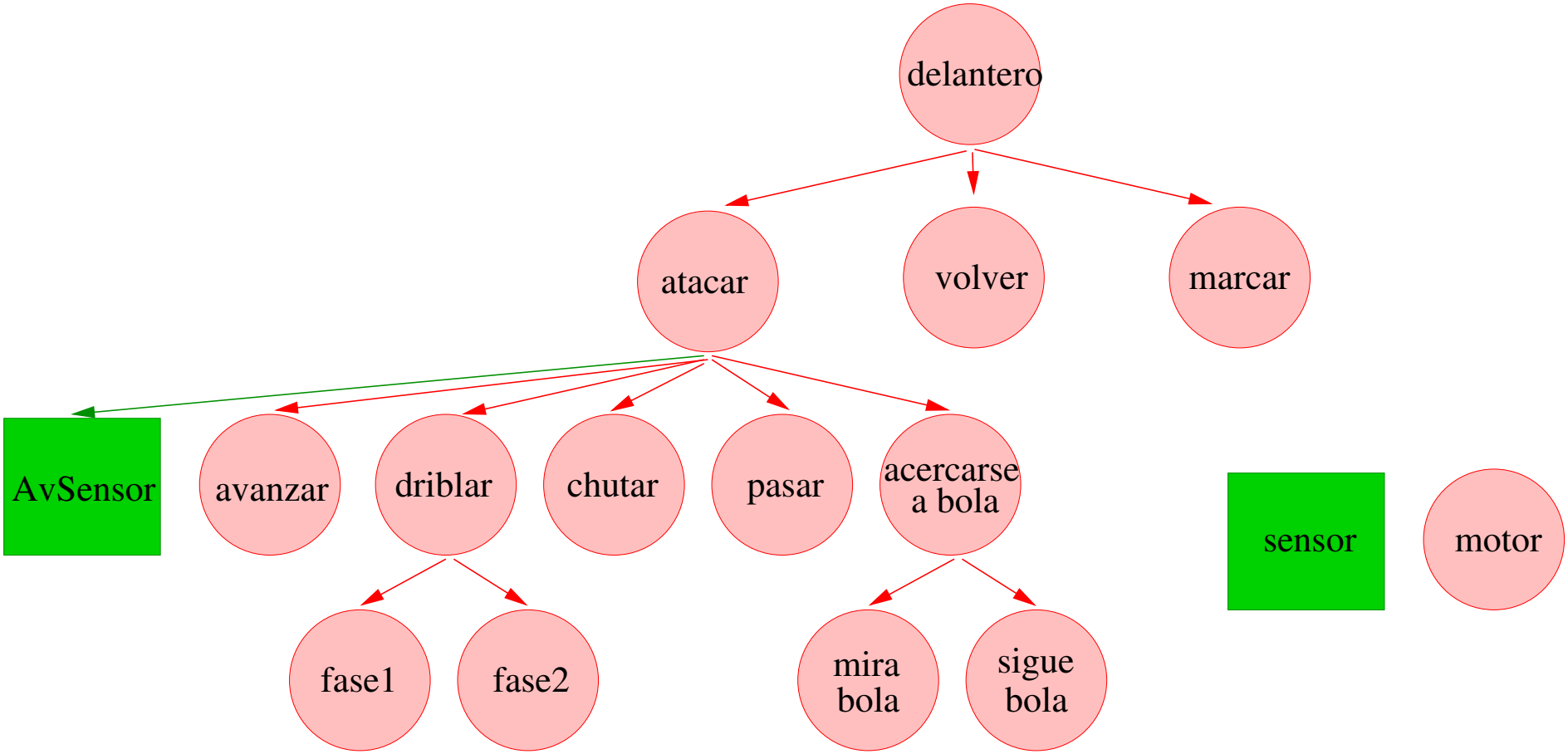
  Perceptivos : dormido, activo

  Motrices : dormido, activo, alerta y preparado

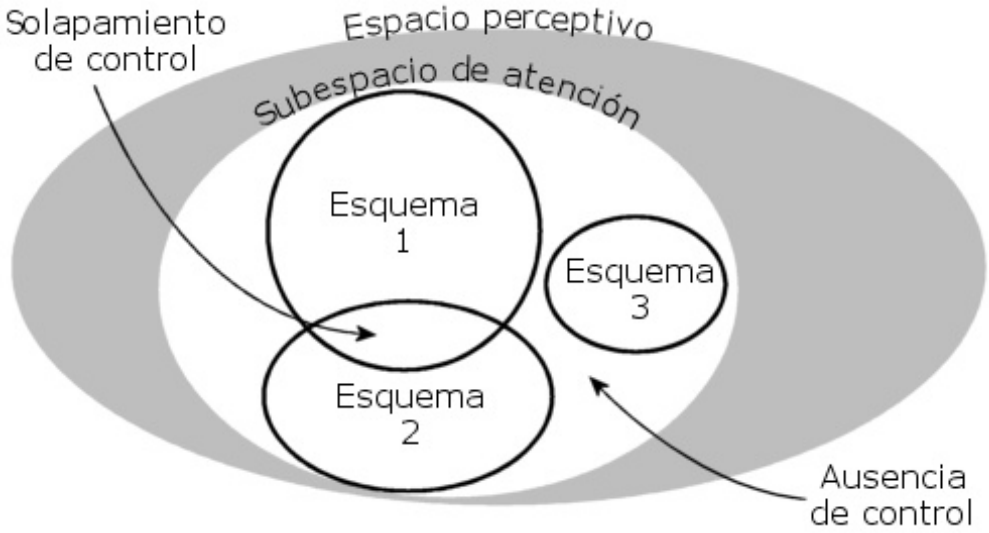
Ejemplo de esquema: MiraBola



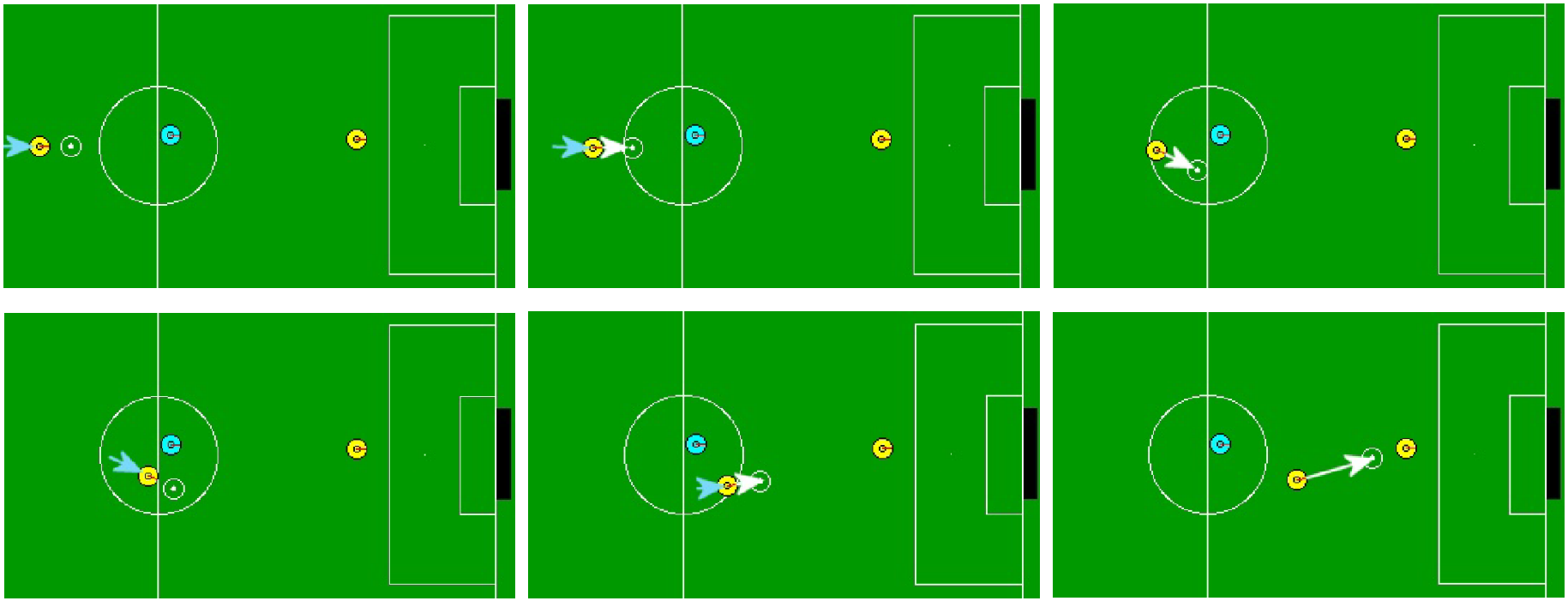
Comportamientos de delantero



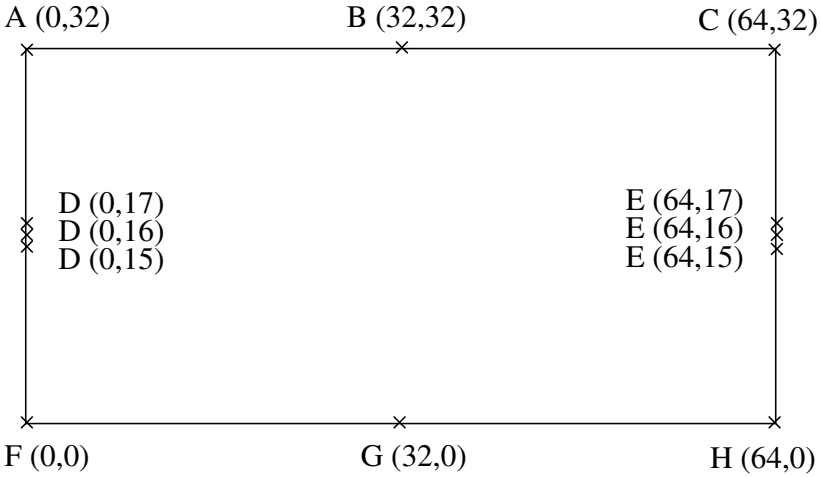
Espacio perceptivo: regiones de activación

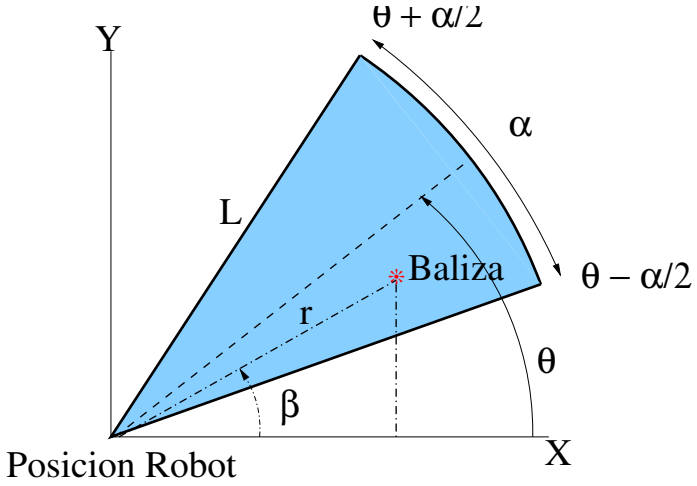


Ejemplo

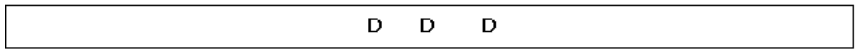


Localización probabilística con visión



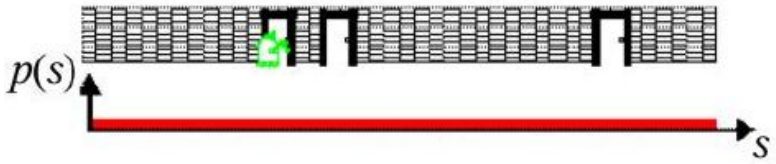


- Mapa: campo de la RoboCup
- Simulación con ruido de la cámara



- Modelo de actuación con ruido y simulación de movimientos
- Próximos a robots reales: EyeBot, Aibo

Localización Bayesiana



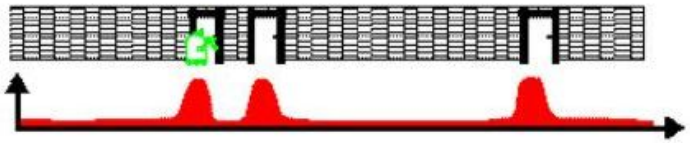
- Bayes para fusionar evidencias:

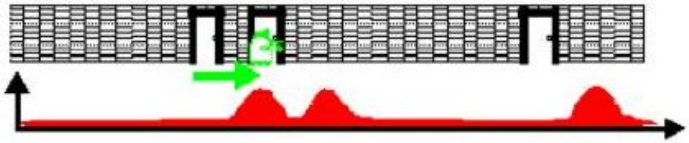
  $\rho(C_{xy\theta}(t)) = \frac{\rho_{obs}}{\rho_{apriori}} * \rho(C_{xy\theta}(t-1))$

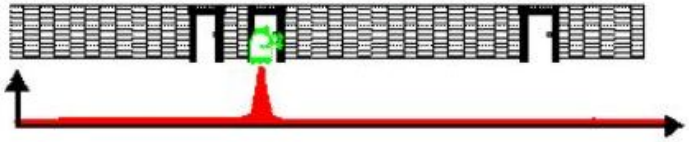
- Modelo probabilístico de observaciones:

  $p(C_{xy\theta}/obs) = e^{-d(obs,teorica)^2}$

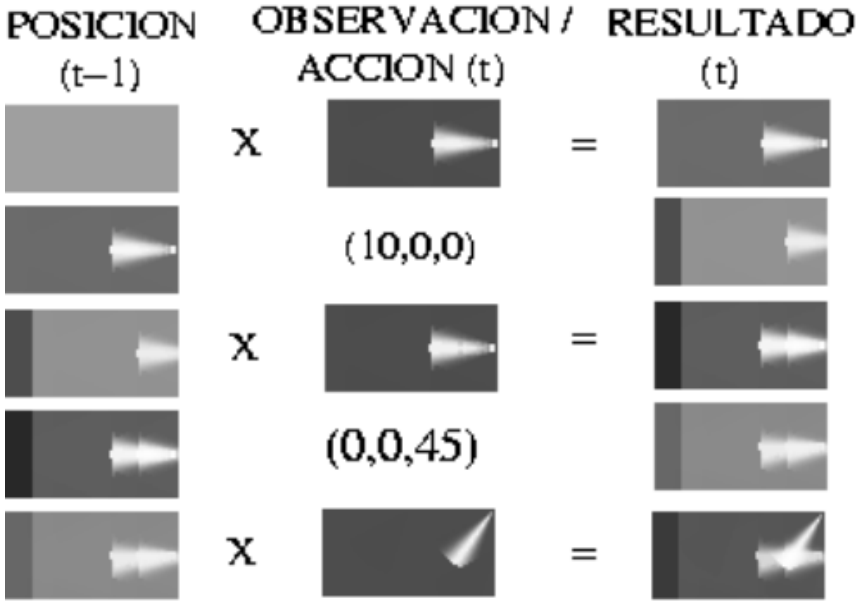
- Distancia entre imágenes teórica y real







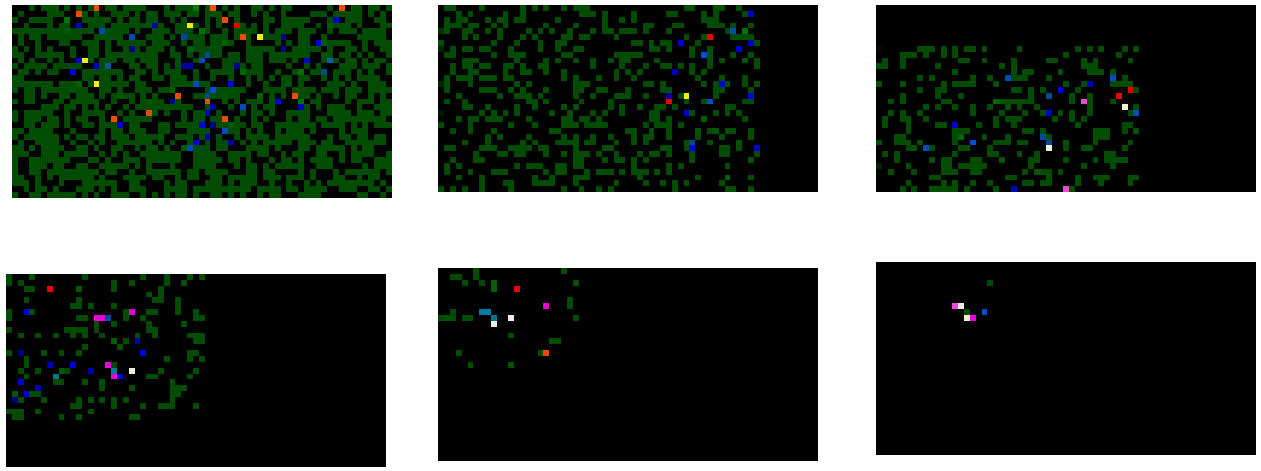
Ejecución típica



- 3 iteraciones, 24s., Pentium III - 1.1GHz
- $33x65x360 = 772,200$ celdillas
- Discretización celdas $1cmx1cm$
- Probabilidad truncada para evitar bloqueos
- Robusto con ruido sensorial y de actuación

- Ventajas:
  1. Adecuada para la localización desconociendo la posición inicial.
  2. Tolera bien incertidumbre en acciones y observaciones.
  3. Permite representar situaciones ambiguas y resolverlas (simetría).
- Desventajas:
  1. No escalable a grandes entornos.
  2. Requiere discretización del espacio.

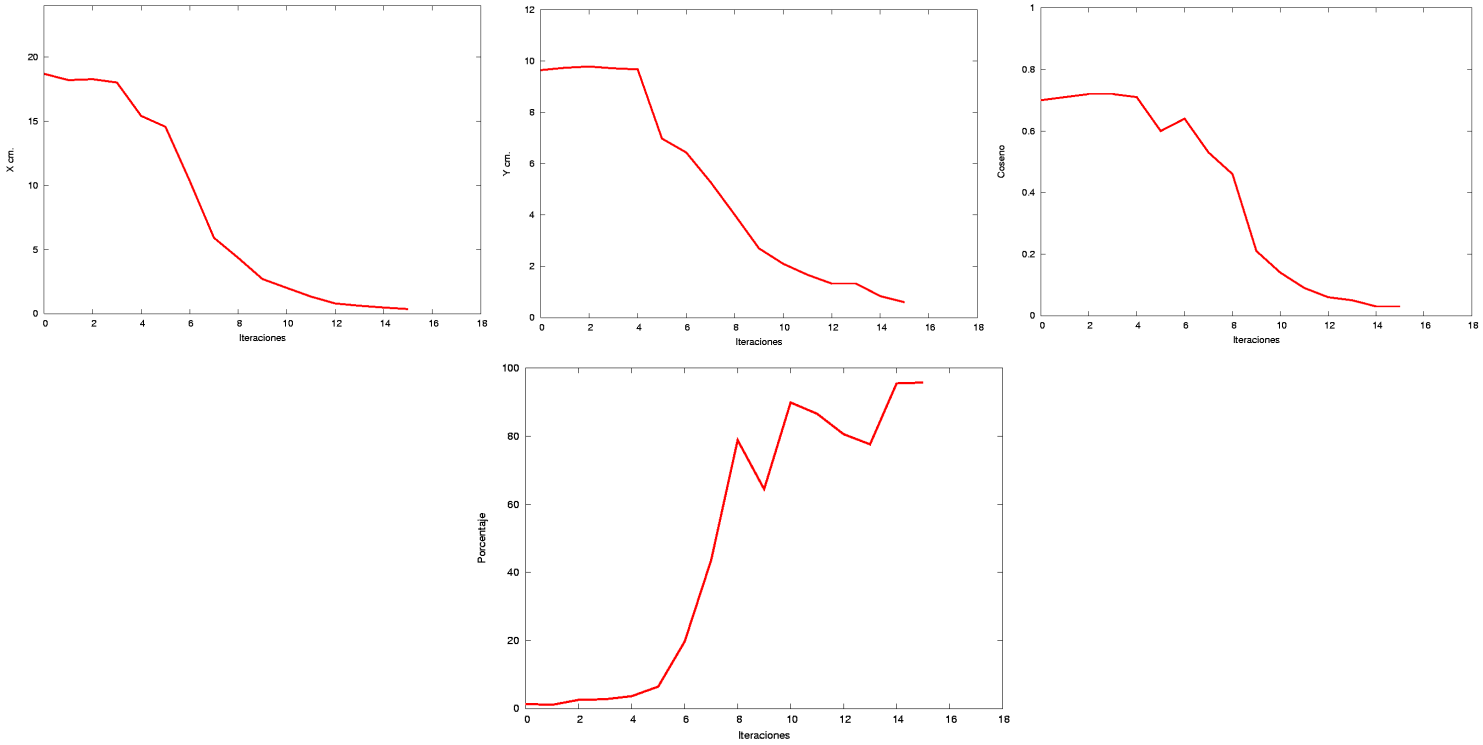
Muestreo: algoritmo CONDENSATION



Población de muestras con probabilidad asociada

1. Predicción: proyección de movimiento, con *"error"*
2. Actualización: modelo de observaciones, $p(C_{xy\theta}/obs) = e^{-d(obs,teorica)^2/256}$
3. Remuestreo: muestras más probables pasan a la siguiente población

Ejecución típica



- Converge en 13-16 iteraciones, 4s., Pentium-III - 1.1 GHz
- 2,000 muestras, precisión 1.9cm
- Sensible al ruido y parámetros internos

- Ventajas:
  1. Permite agilizar los cómputos (número acotado de muestras).
  2. Escalable a grandes entornos.
  3. Resultados continuos.
- Desventajas:
  1. Requiere un ajuste muy fino de todos sus parámetros.

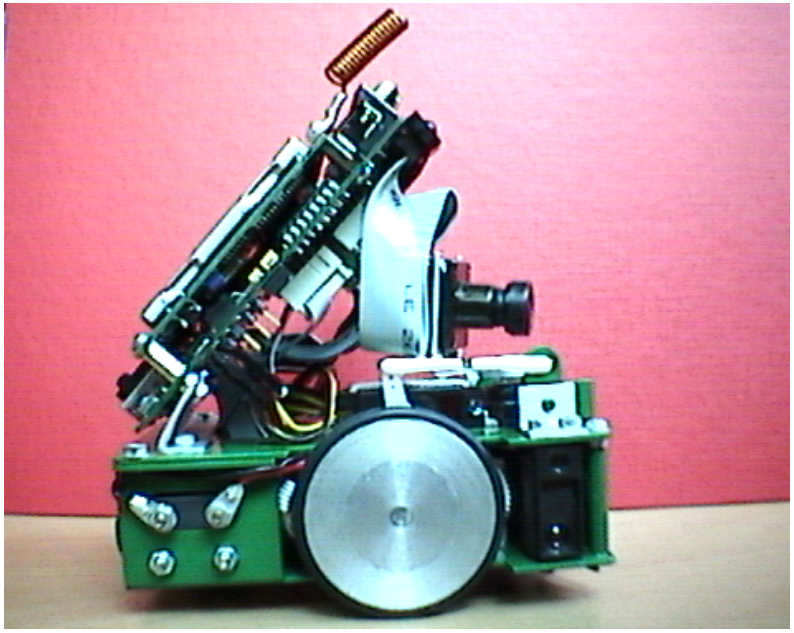
EyeBot

Indice

- hardware
- programación
- teleoperador
- sigue-pelota local
- sigue-pelota cenital
- sigue-pared

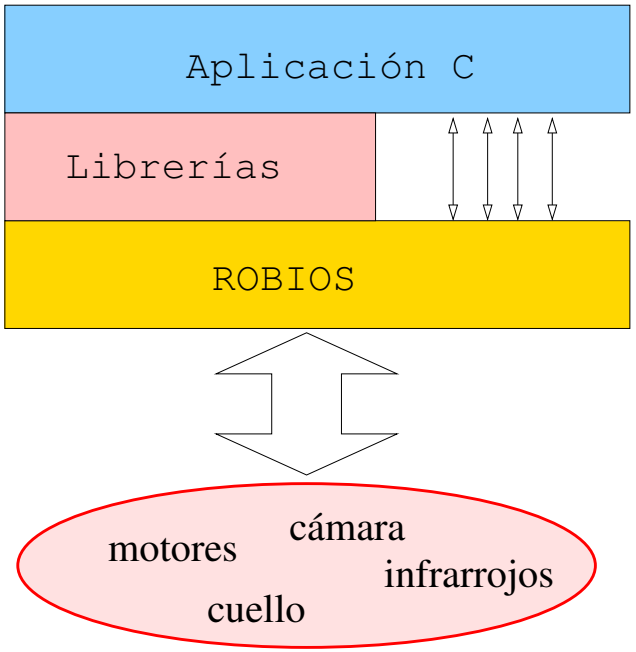
Hardware del robot EyeBot

- 2 motores
- 2 servos
- 2 odómetros
- 3 sensores infrarrojo
- 1 cámara (83 x 64 pixels)
- motorola 68332, 35 Mhz
- enlace radio, 9600 bps
- pantalla B&W y botones

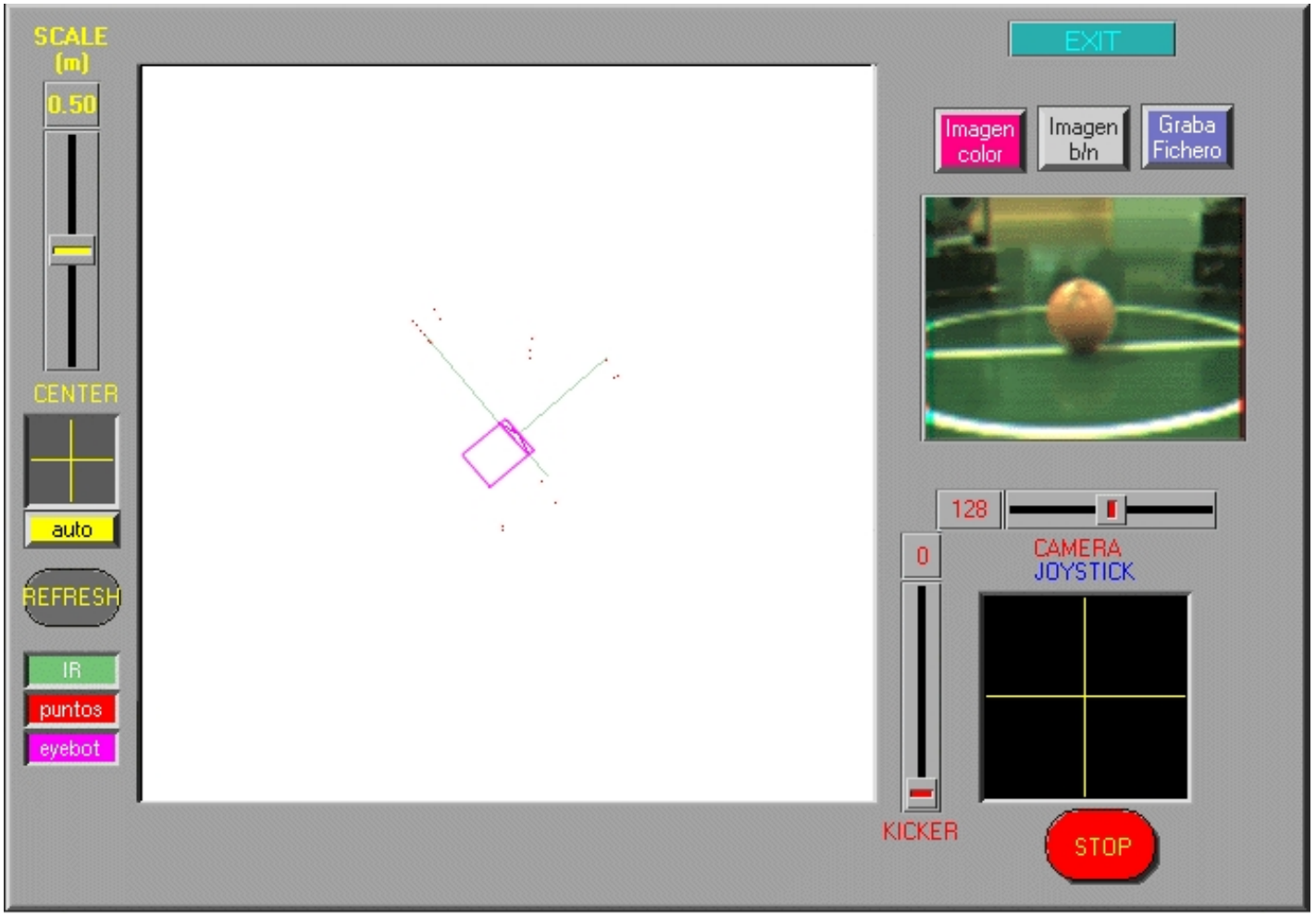


Programación EyeBot

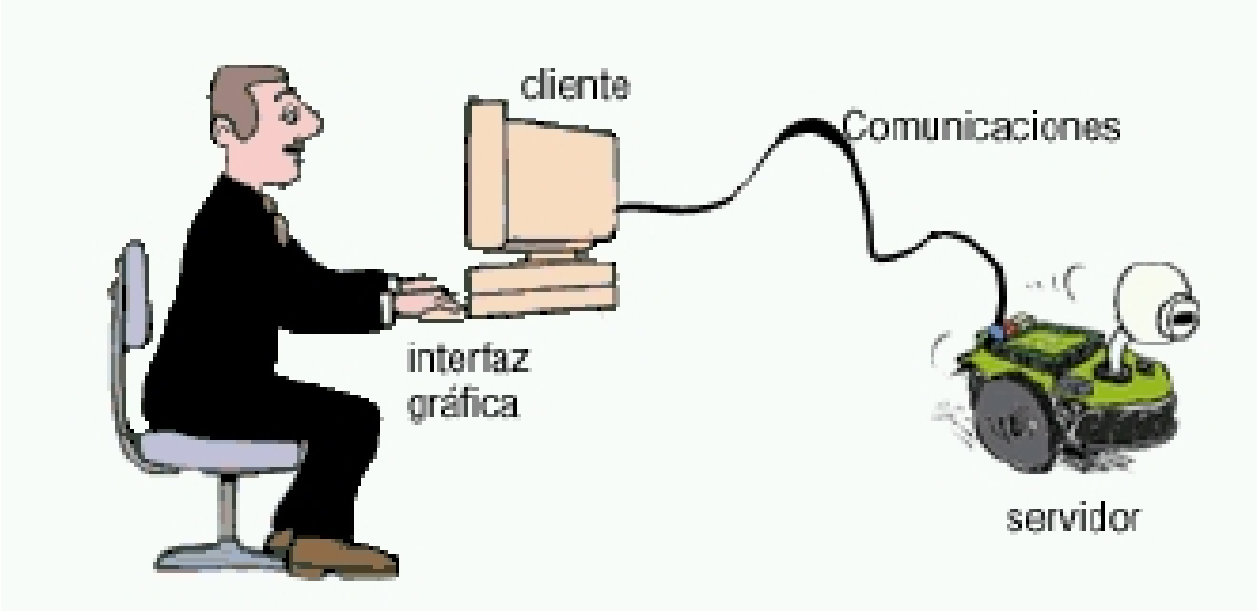
- compilador cruzado en el PC
- descarga del ejecutable por puerto serie
- RoBios: sistema operativo del robot, API
- Tabla de descripción del hardware (HDT)
- librerías: V-W, filtros imágenes



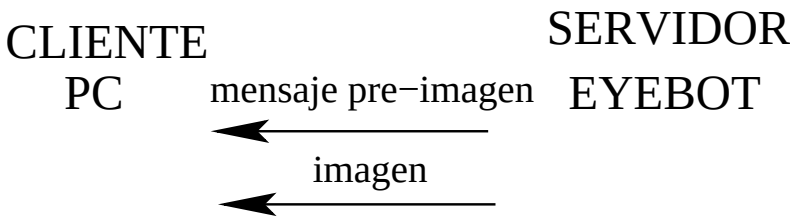
Teleoperador



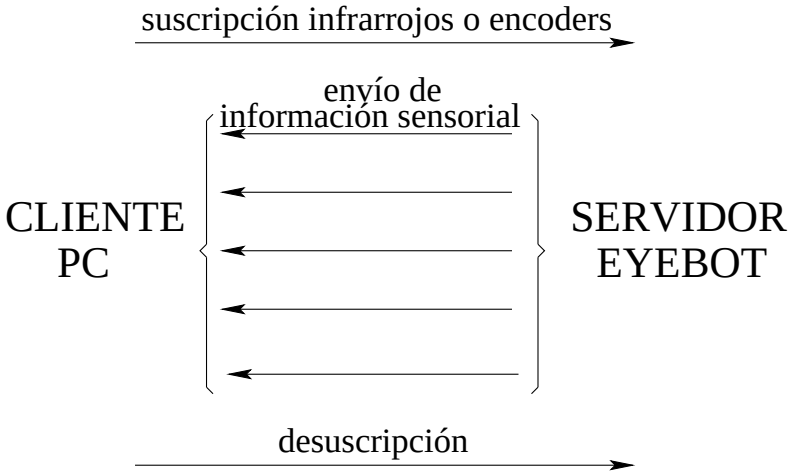
diseño



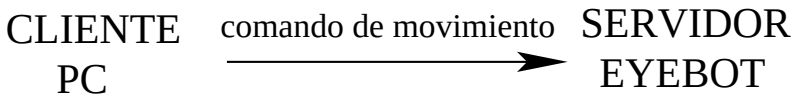
protocolo



Transmisión de imágenes

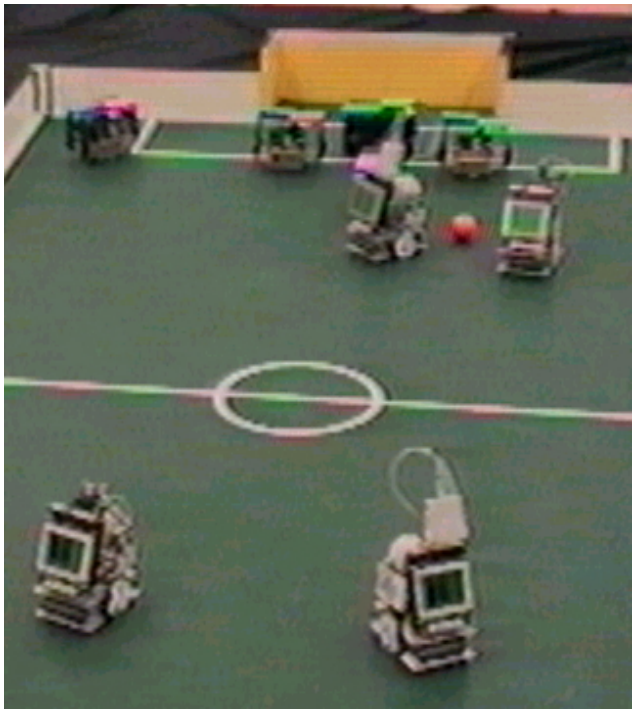


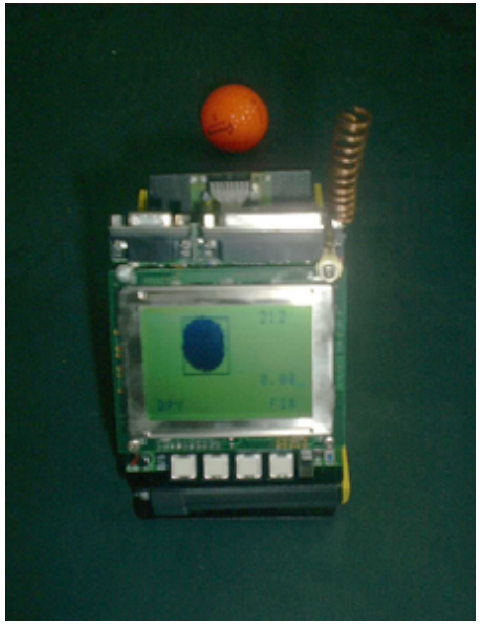
Envíos por suscripción / desuscripción



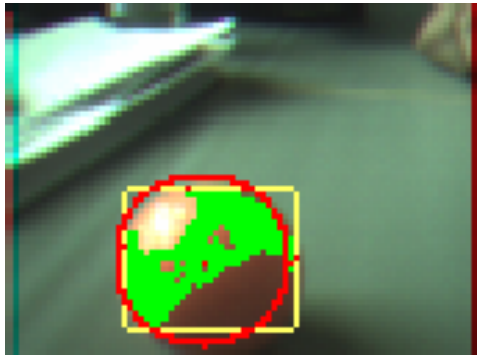
Envío de comandos de movimiento

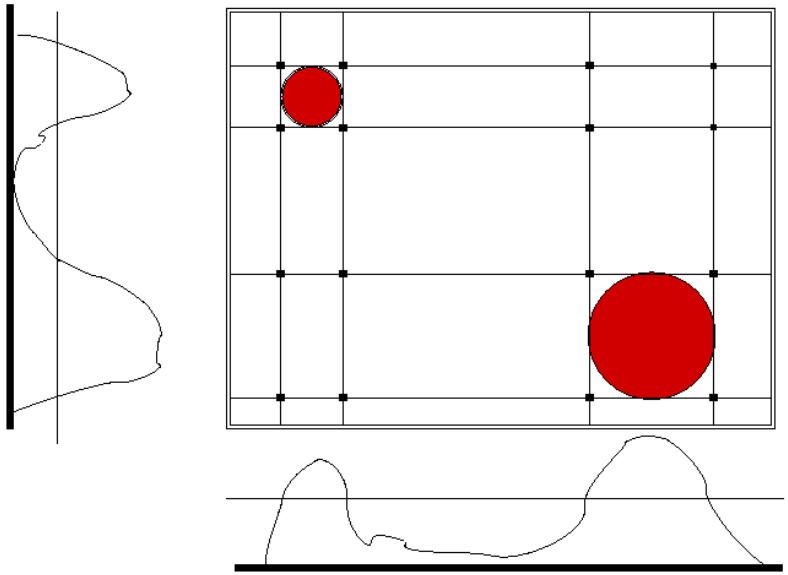
Sigue-pelota local



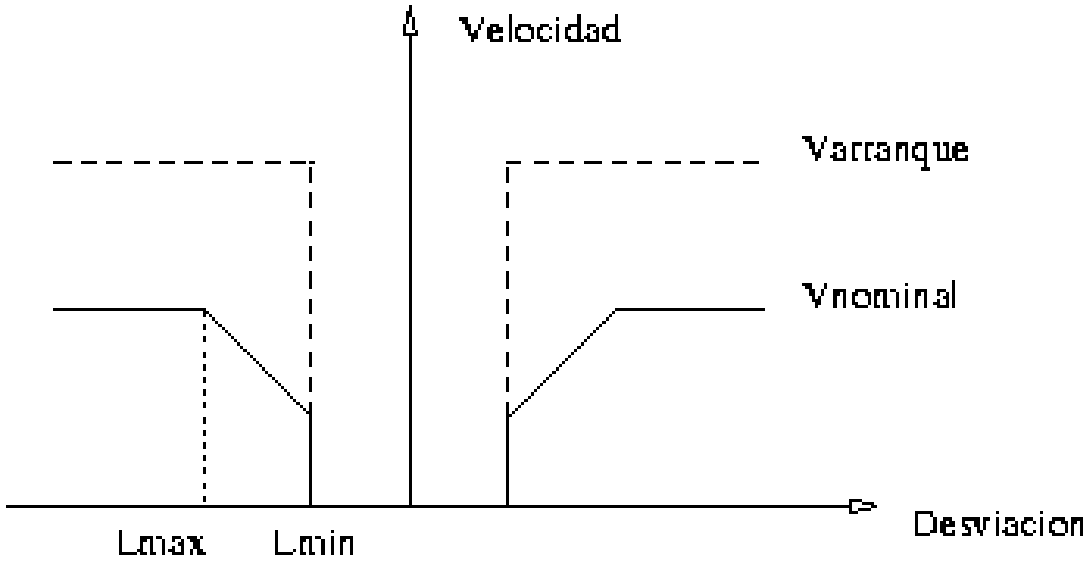


Percepción



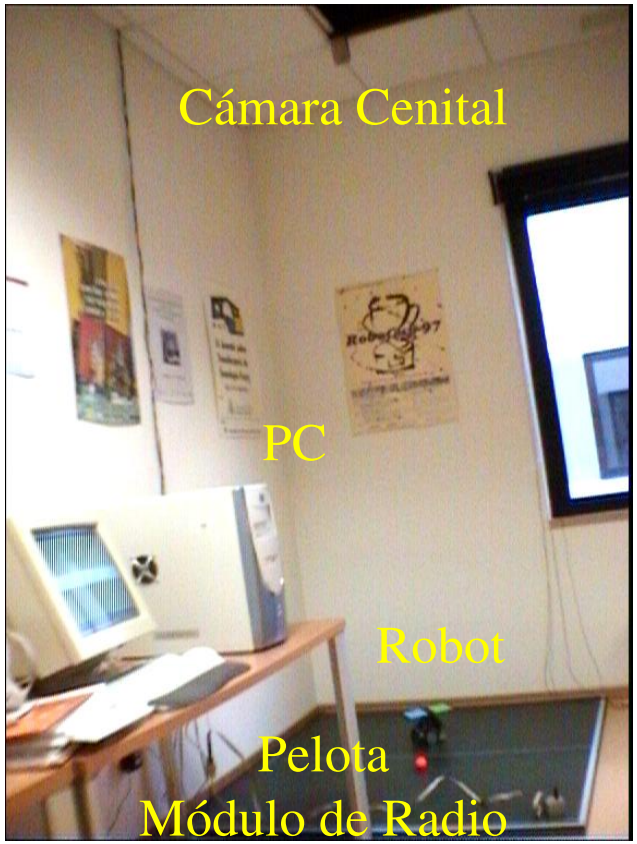


Control proporcional en velocidad



- centro de masas pelota
- pico de arranque
- 3 fps

Sigue-pelota cenital



- visión cenital
- etiquetas color
- video4linux
- 9 fps
- radio

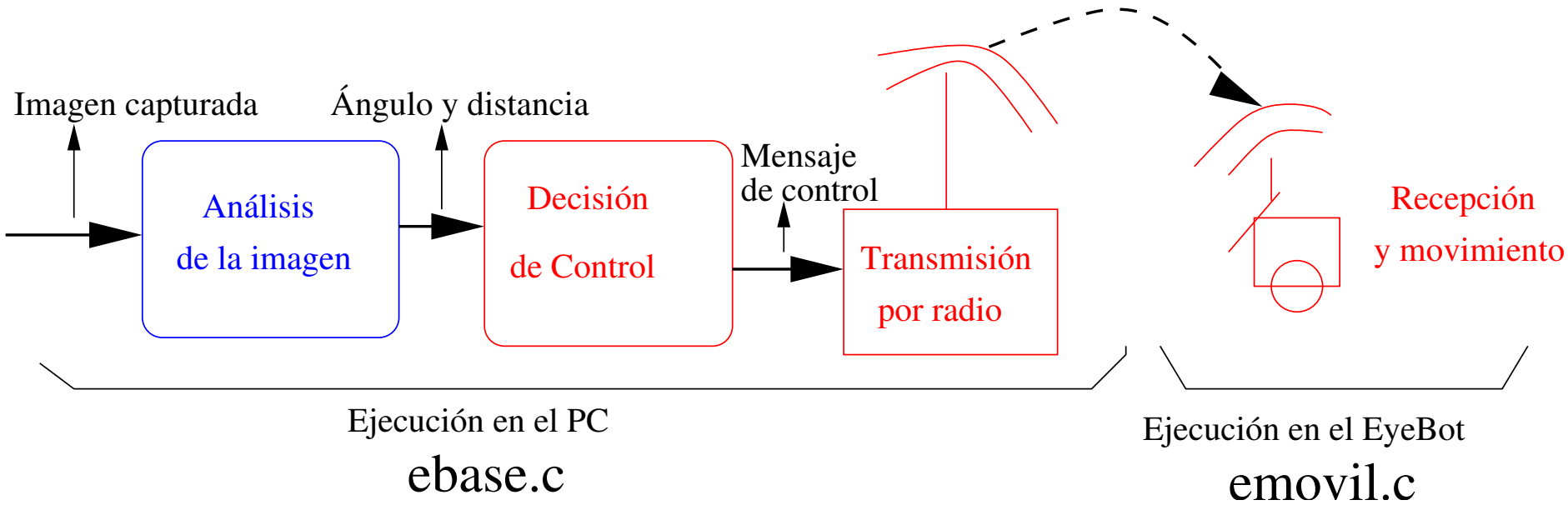
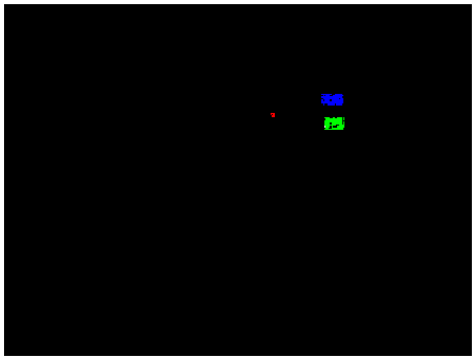


Imagen capturada
Ángulo y distancia
Análisis
de la imagen
Decisión
de Control
Mensaje
de control
Transmisión
por radio
Recepción
y movimiento
Ejecución en el PC
ebase.c
Ejecución en el EyeBot
emovil.c

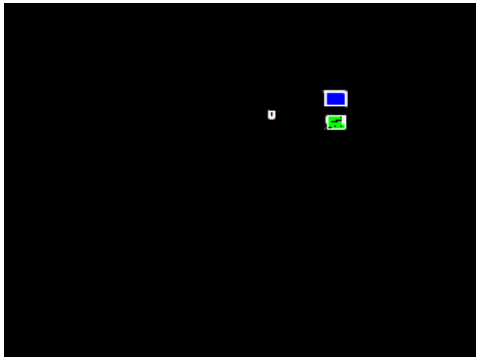
Percepción

inicial

filtrada

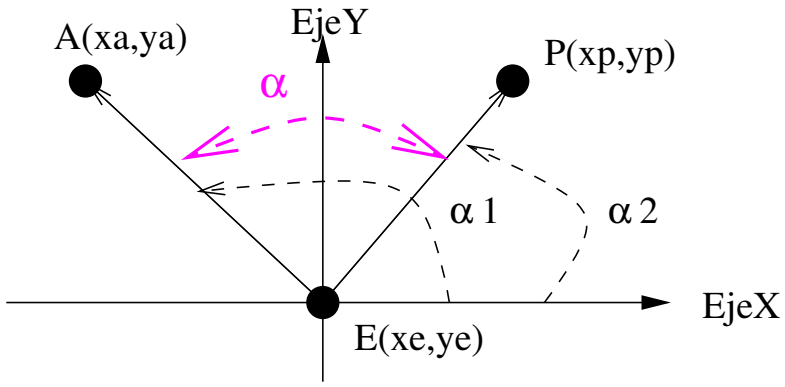


segmentada



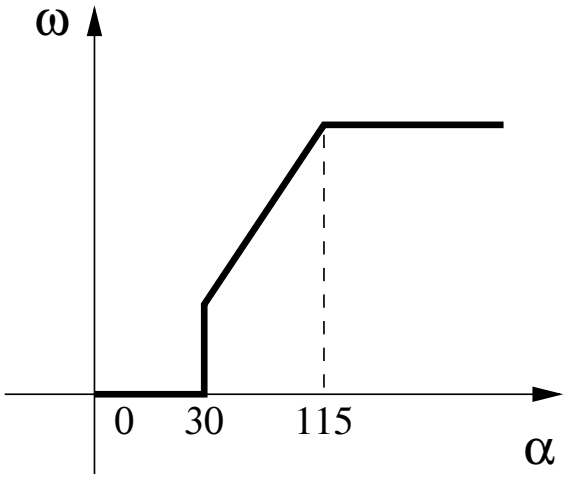
Identificación de la pelota y del EyeBot

- Tres puntos: $P(x_p, y_p)$, $A(x_a, y_a)$, $E(\frac{x_A+y_V}{2}, \frac{y_A+y_V}{2})$
- $d = \left|\vec{EP}\right| = \sqrt{x_{EP}^2 + y_{EP}^2}$

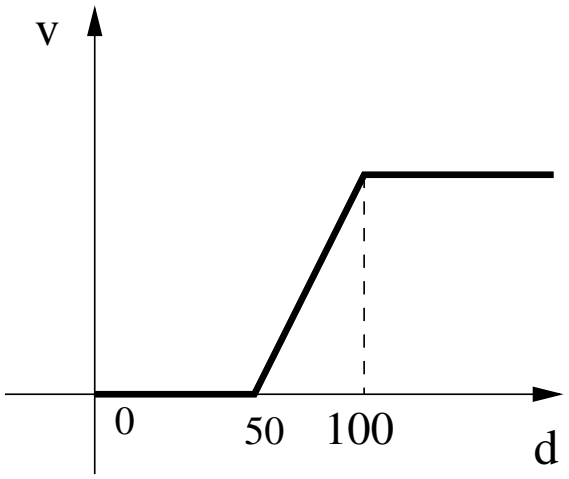


Control en velocidad

En función de α y d tomamos las decisiones de moviento que se materializará en un control de velocidad del robot (v y ω).



Velocidad Angular vs Ángulo

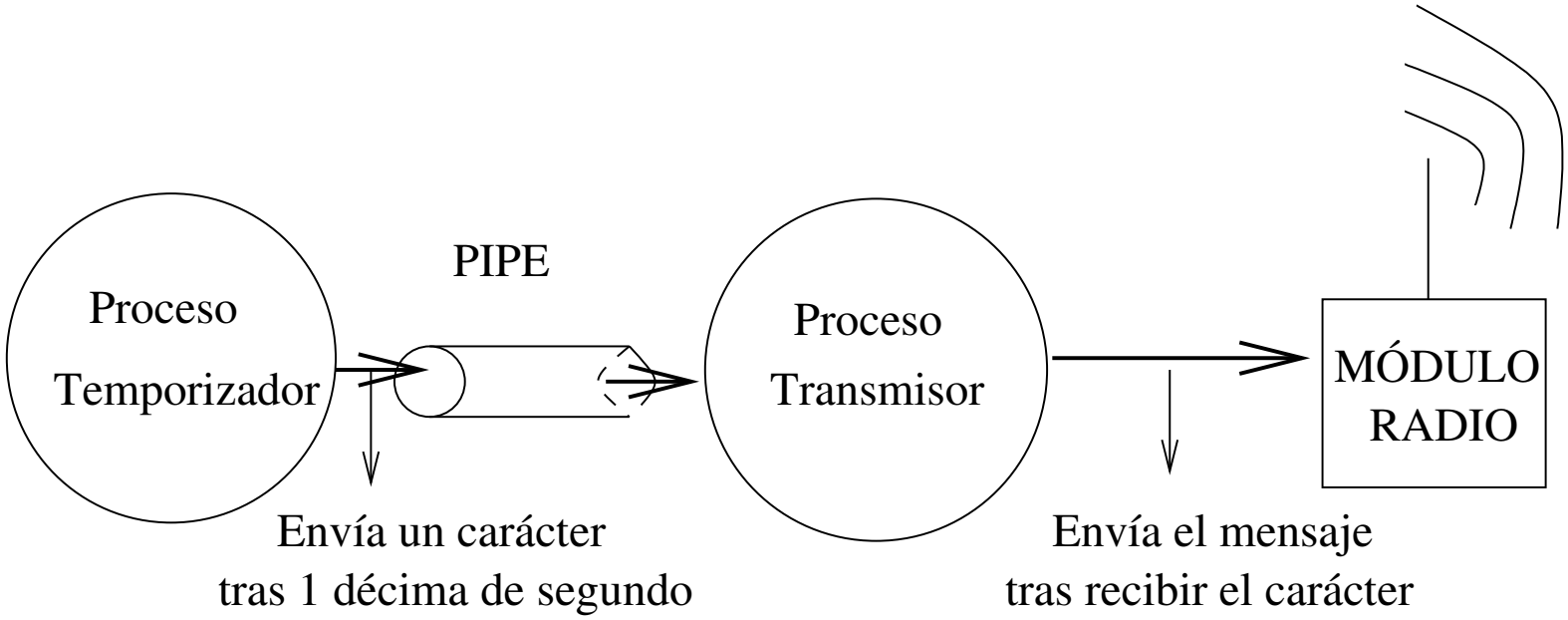


Velocidad Lineal vs Distancia para 0<= α <=30

Transmisión por radio

COMMANDED	0.1	1.0

- Saturación del Búffer de envío
- Señal SIGALARM

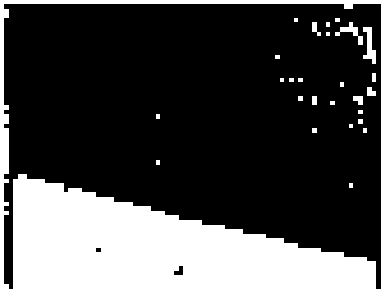


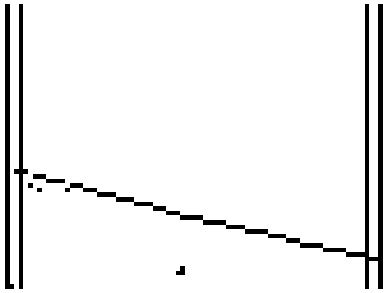
Sigue-pared

Filtro de color

- en espacio RGB
- ajuste manual

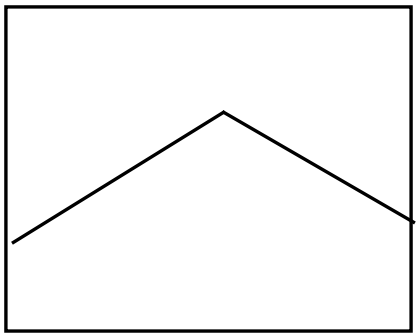
Bordes



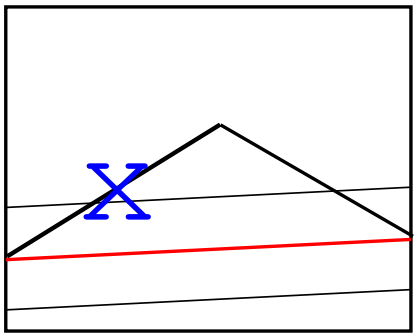


- de abajo a arriba
- robusto

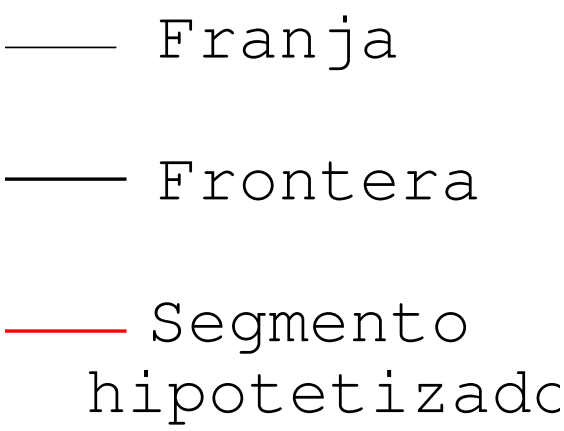
Segmentación

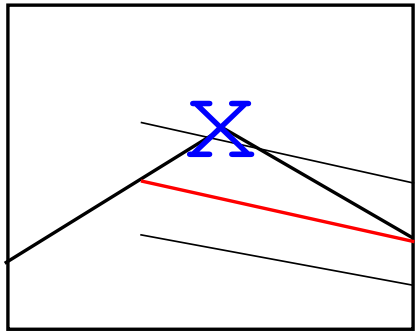


Comienzo de la segmentación

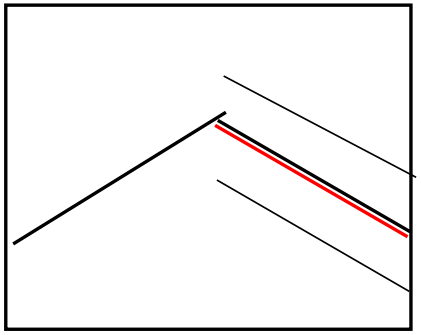


Pixel fuera de franja

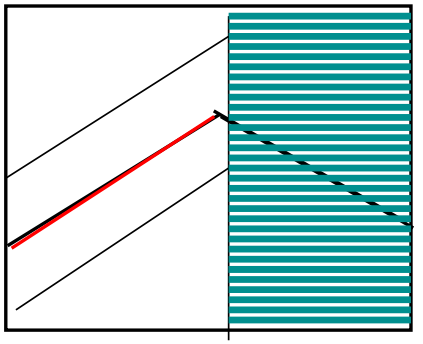




Pixel fuera de franja



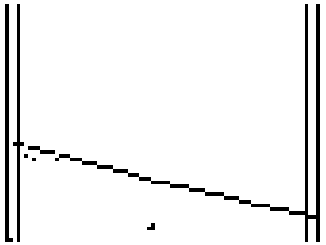
Verificación de hipótesis



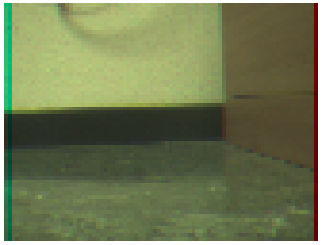
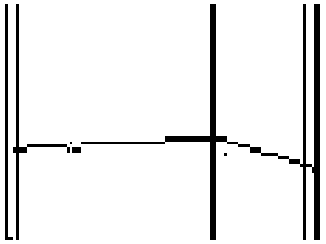
Explicar resto de frontera

Control basado en casos

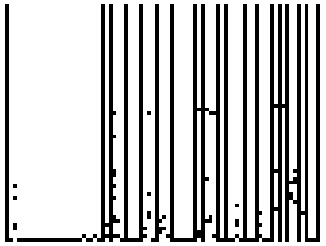
- Pared infinita.

- Esquina cóncava.

- Discontinuidad espacial.

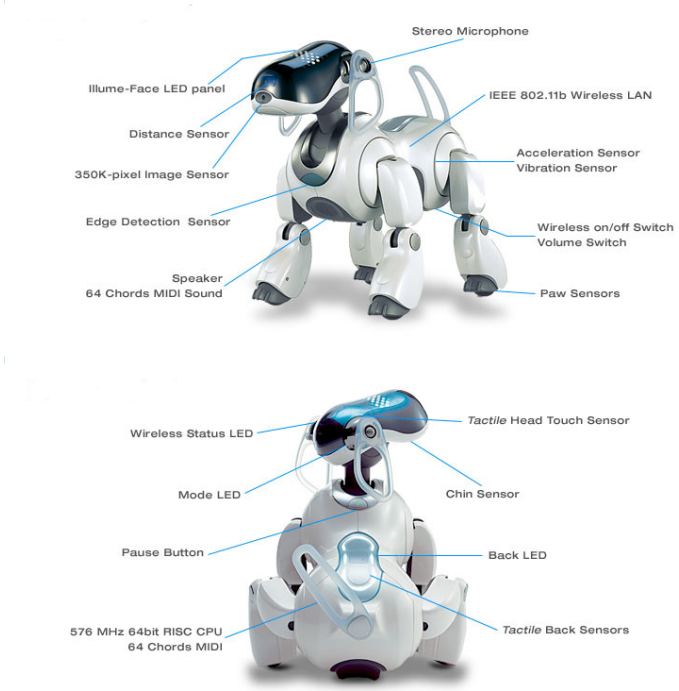
- Todo suelo, todo pared

Programación del robot Aibo

Indice

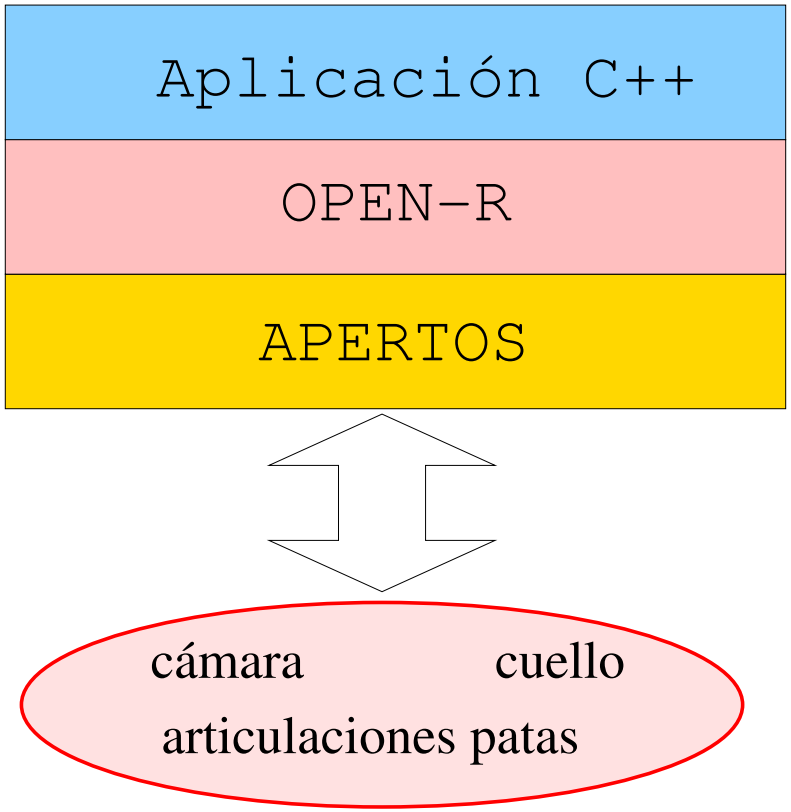
- hardware robot Aibo
- entorno de programación
- ejemplo: teleoperador
- ejemplo: sigue-persona

Hardware del AIBO ERS-7



- Procesador MIPS 576 Mhz, 64 MB RAM.
- Cámara color 350 K-pixels
- 20 grados de libertad.
- 14 sensores
- Tarjeta wireless 802.11b

Arquitectura software



Sistema Operativo Apertos

- Sistema Operativo empotrado
- Basado en Objetos
  - El hardware es modelado como un conjunto de objetos
  - Las aplicaciones son objetos o conjunto de objetos cooperantes
- Los objetos son concurrentes entre sí y tienen un solo hilo de ejecución
- Comunicación entre objetos por paso de mensajes
- Un objeto no puede acceder al espacio de direcciones de otro objeto, pero no hay restricciones de acceso a la memoria compartida

OPEN-R

- Interfaz "sencillo" de programación que proporciona Sony
- Más fácil que programar directamente encima de Apertos
- Existen interfaces superiores: Tekkotsu, R-Code, AIBO Remote Framework, Aibo Motion Editor...
- objetos básicos para acceder al hardware
  - OVirtualRobotComm
  - OVirtualRobotAudioComm
  - ANT
- Relativamente reciente

Aplicaciones en OPEN-R

- Orientación a objetos
  - Un binario por objeto
  - Esqueleto fijo: DoInit, DoStart, DoStop, DoDestroy
- Orientación a eventos, puntos de entrada
  - por eventos de mensajes: *Notify* y *Ready*
- Comunicación asíncrona entre objetos a través de mensajes
  - roles: *Observer* y *Subjects*

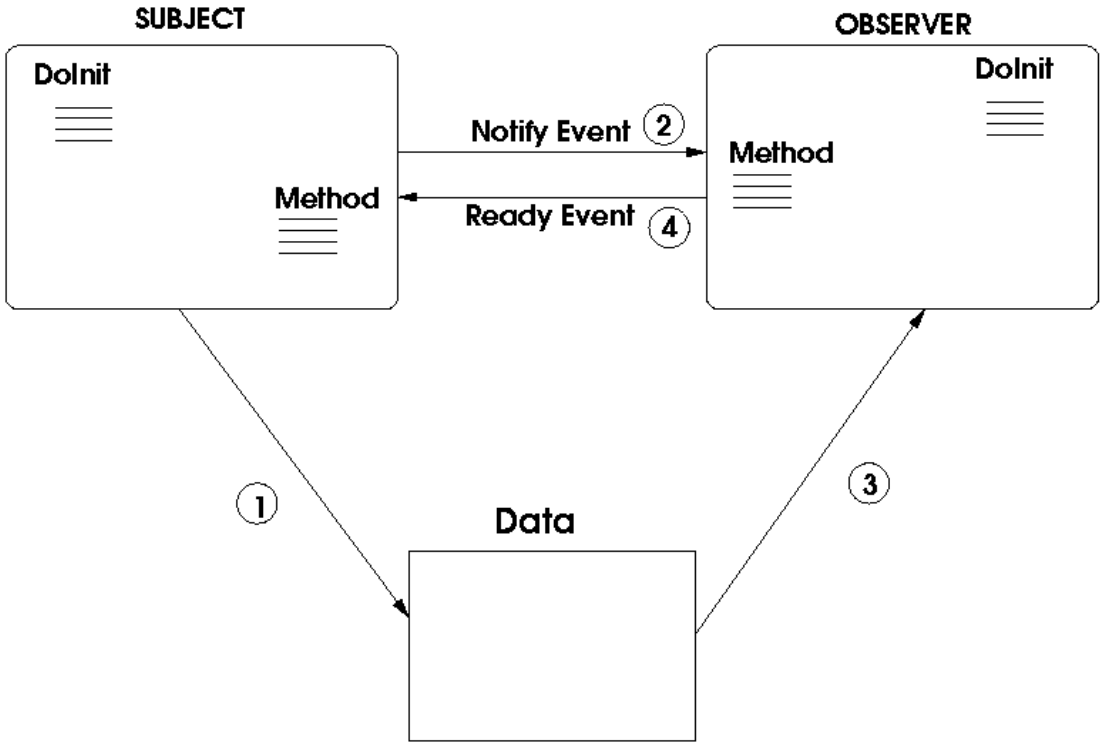
Ejecución concurrente

- Una aplicación se compone de varios objetos se ejecutan de manera concurrente
- Cada objeto tiene un único hilo de ejecución
- Ningún objeto puede sobreescribir la memoria de otro
- Acceso a memoria compartida mediante mecanismos de exclusión mutua para evitar condiciones de carrera

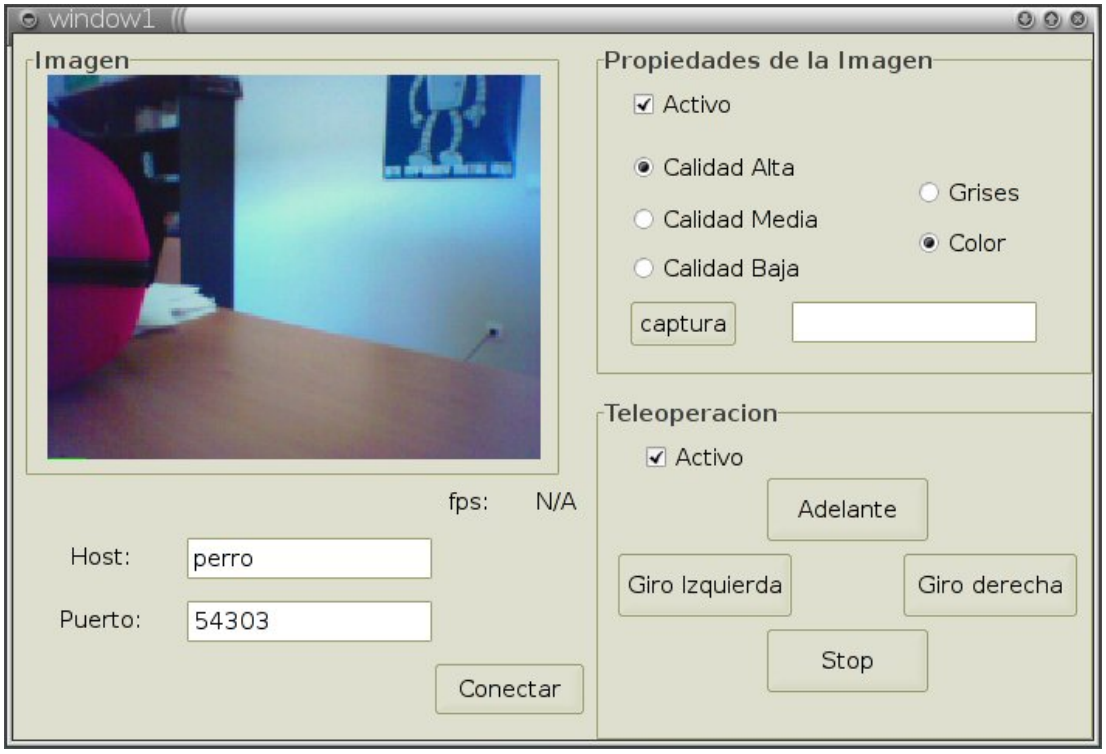
Comunicación y sincronización entre objetos

- Configuración por fichero, `connect.cfg`
  - tipo de los mensajes
  - de las subscripciones Subject-Observer mediante pares de puertos
- Configuración por objeto, `stub`
  - de los puertos de un objeto con su manejador
- Cuando llega un mensaje al puerto de un objeto:
  - Se encola en un único buffer por objeto donde se almacenan los mensajes que aún no han sido atendidos
  - Cuando se atiende se invoca el manejador

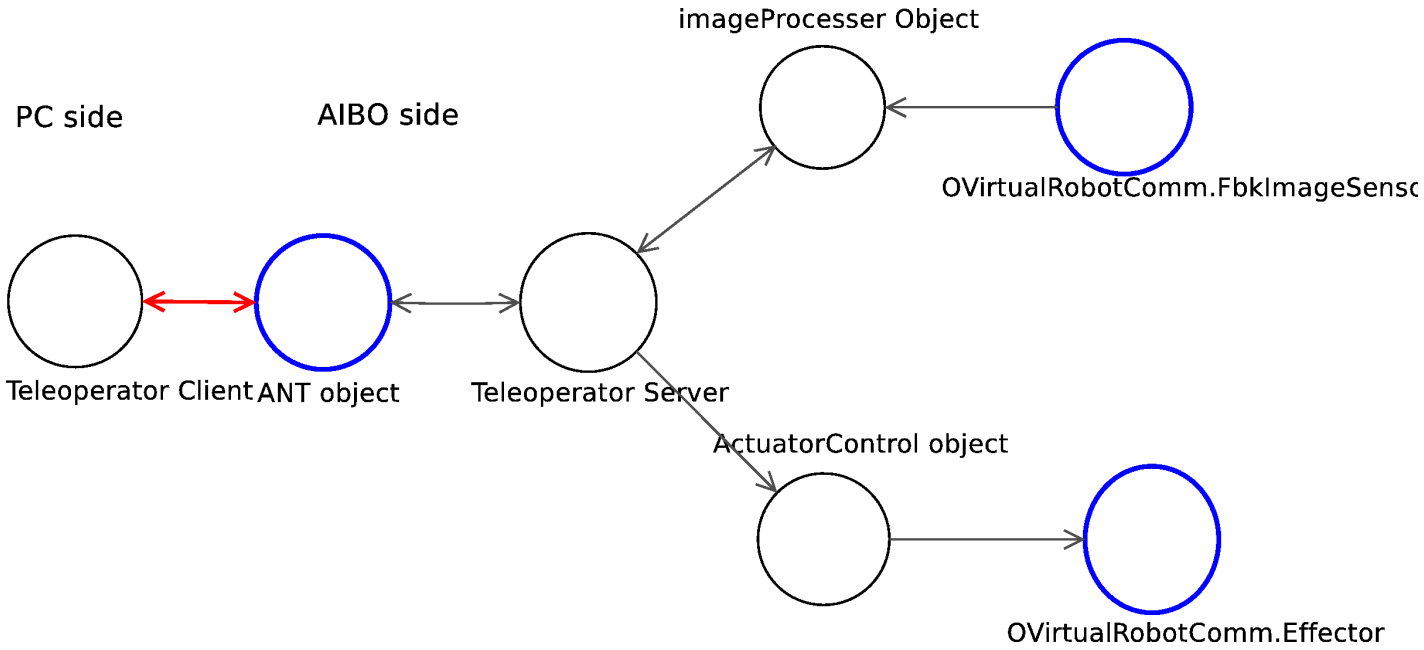
- La sincronización entre objetos se realiza con mensajes
- Los mensajes Ready que se está preparado para recibir datos
- Los mensajes Notify que tiene datos nuevos en la memoria compartida



Ejemplo: teleoperador



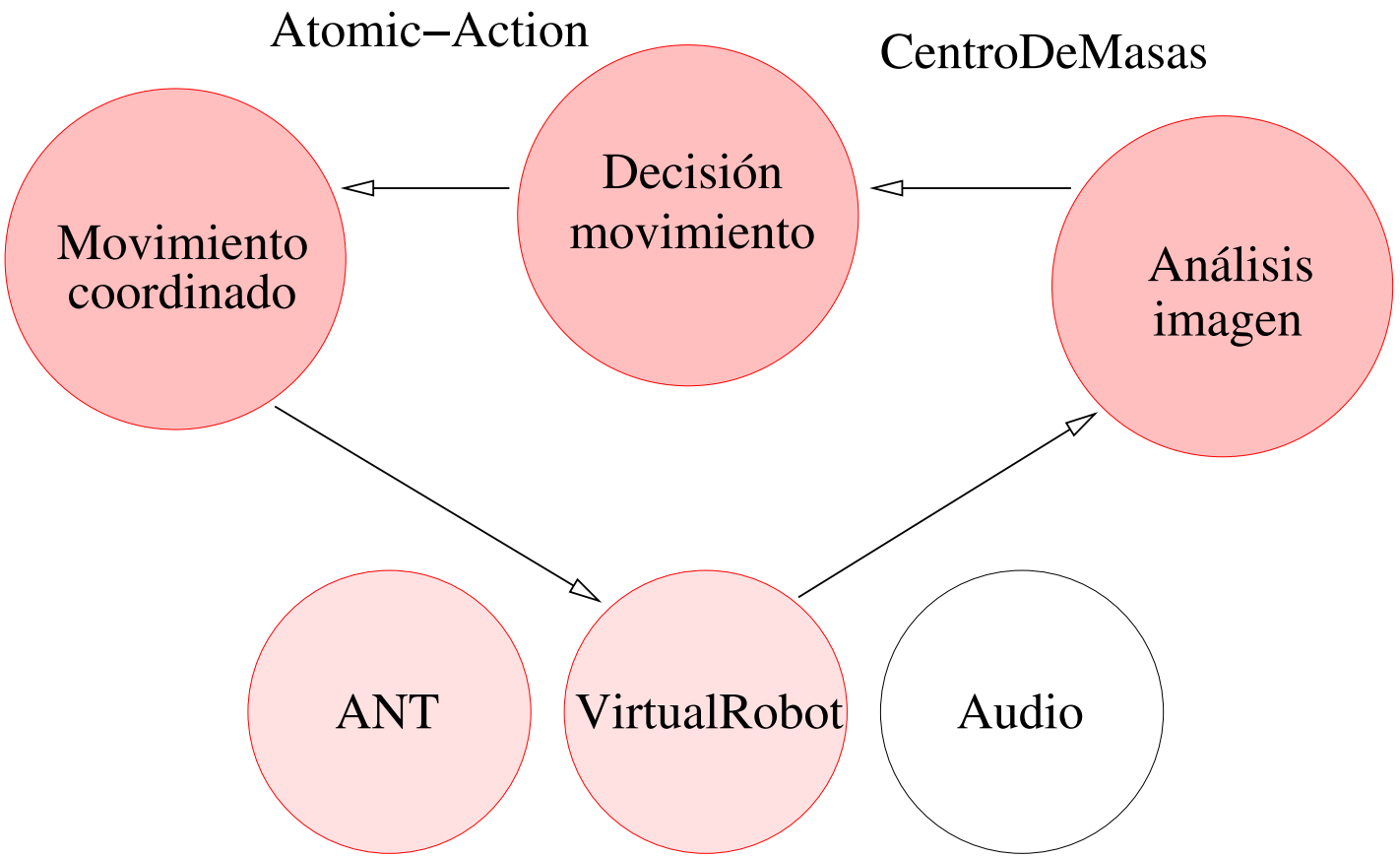
diseño


imageProcesser Object
PC side
AIBO side
OVirtualRobotComm.FbkImageSensc
Teleoperator Client
ANT object
Teleoperator Server
ActuatorControl object
OVirtualRobotComm.Effector

Ejemplo: sigue-persona

- cuello y cuerpo
- filtro de color
- 13-18 fps
- control discreto

diseño


Atomic−Action
CentroDeMasas
Movimiento coordinado
Decisión movimiento
Análisis imagen
ANT
VirtualRobot
Audio