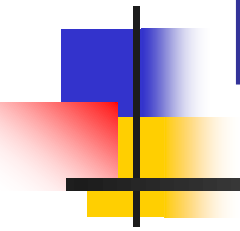
Programación Automática

MÁSTER EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

Ricardo Aler Mur



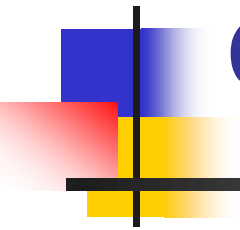
Programación Automática

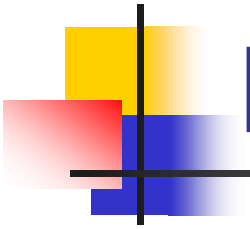
Ricardo Aler Mur

Universidad Carlos III de Madrid

http://www.uc3m.es/uc3m/dpto/INF/aler

Behavioral Cloning (imitación de comportamientos)





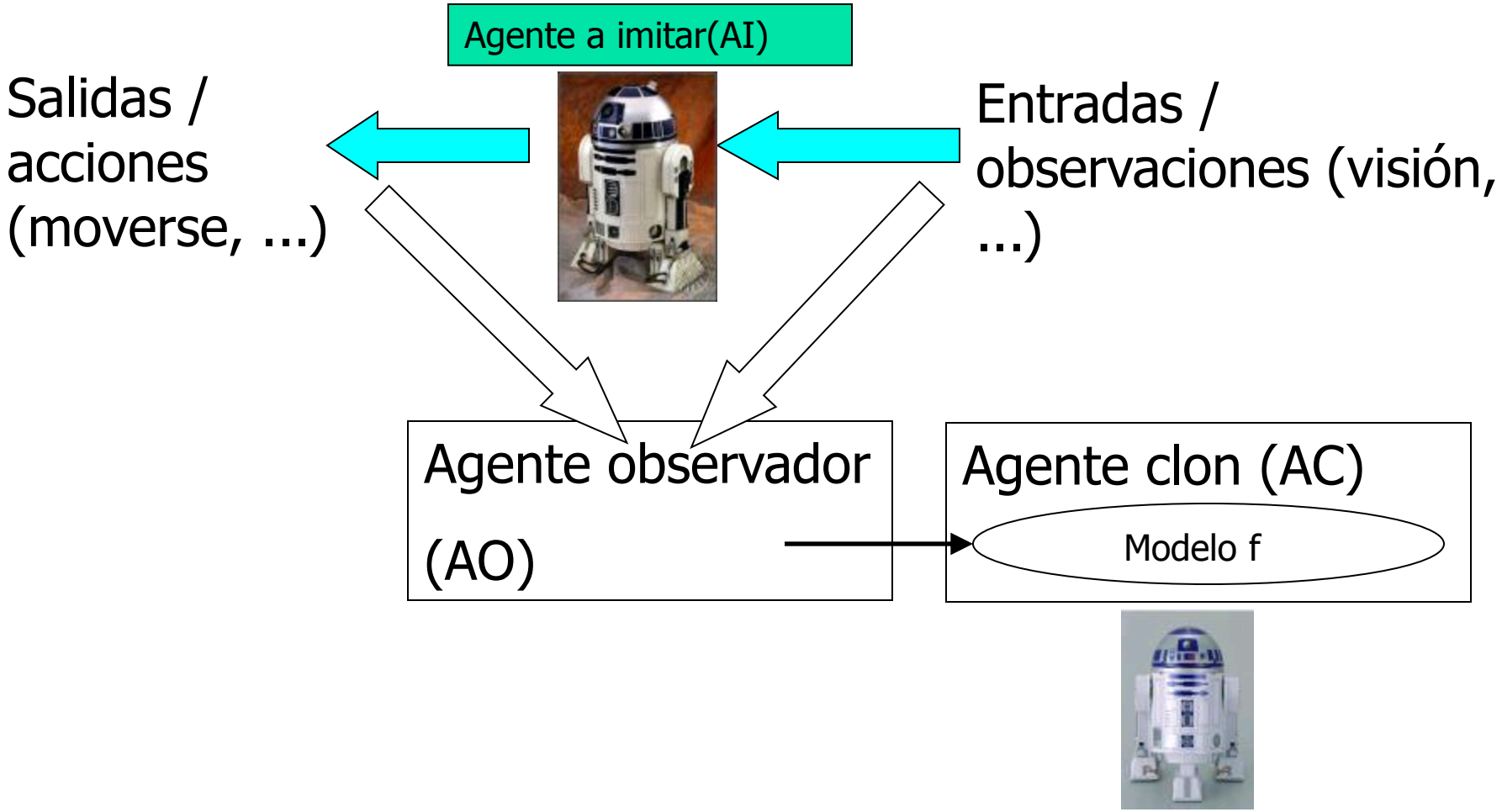
Behavioral Cloning

- Un agente aprende comportamientos de otro agente mediante la observación y la imitación
- El agente a imitar suele ser una persona

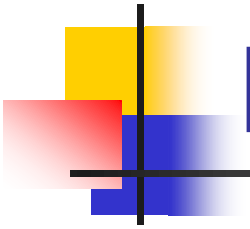
Behavioral Cloning

- Apprenticeship learning; Learning by demonstration; Learning by imitation
- Behavioral cloning is a method by which human subcognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.
- Claude Sammut

Esquema genérico: observación entradas/salidas



Ejemplo de modelo aprendido

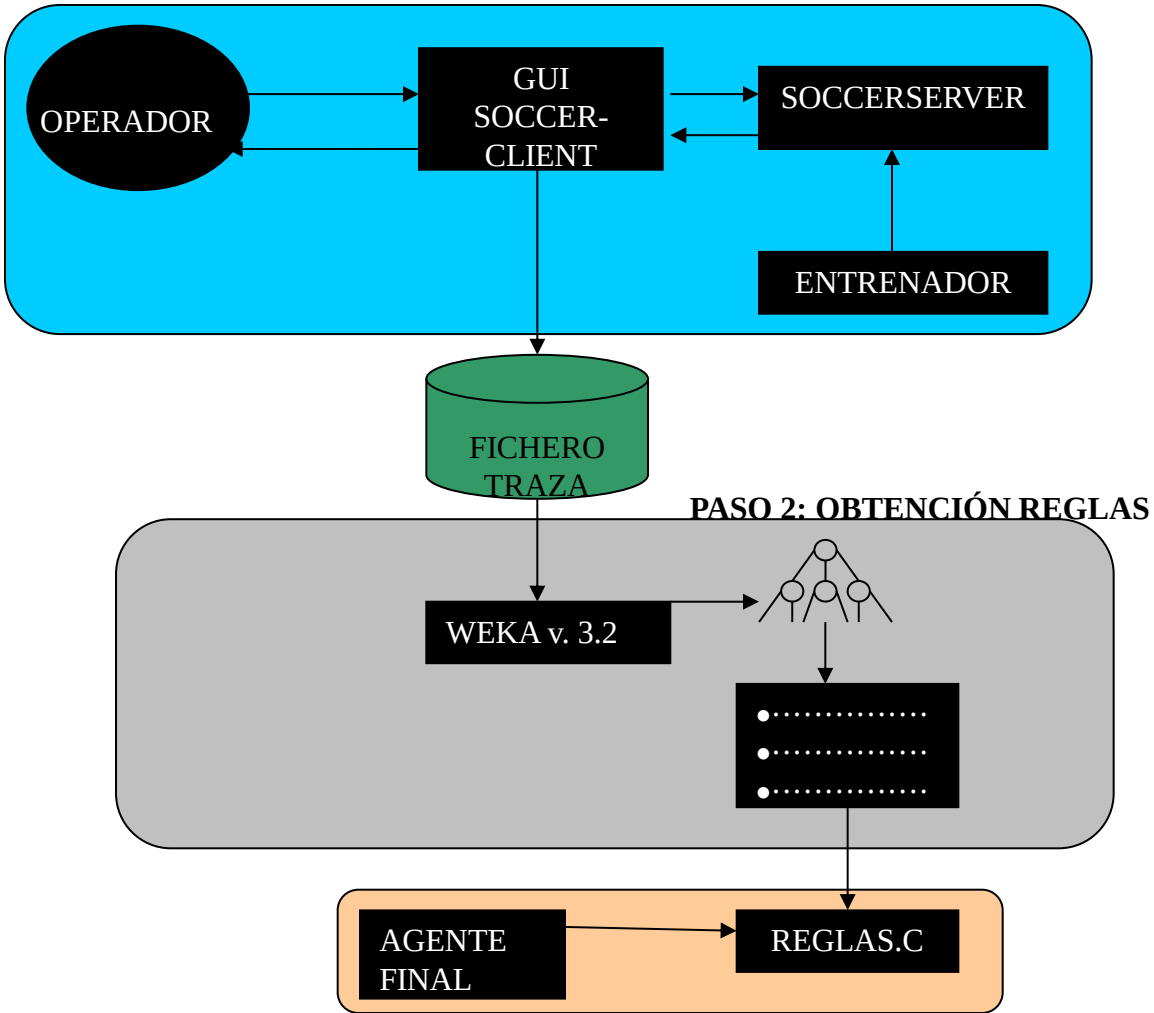


- f: Entradas -> Salidas
- Dada una entrada (sensores) en el instante i, obtener la salida (acción) en el instante i+1
- $f(\text{entrada}_i) = \text{acción}_{i+1}$
- En último término, dadas parejas (entrada, salida) y unos primitivas, determinar el programa: programación automática

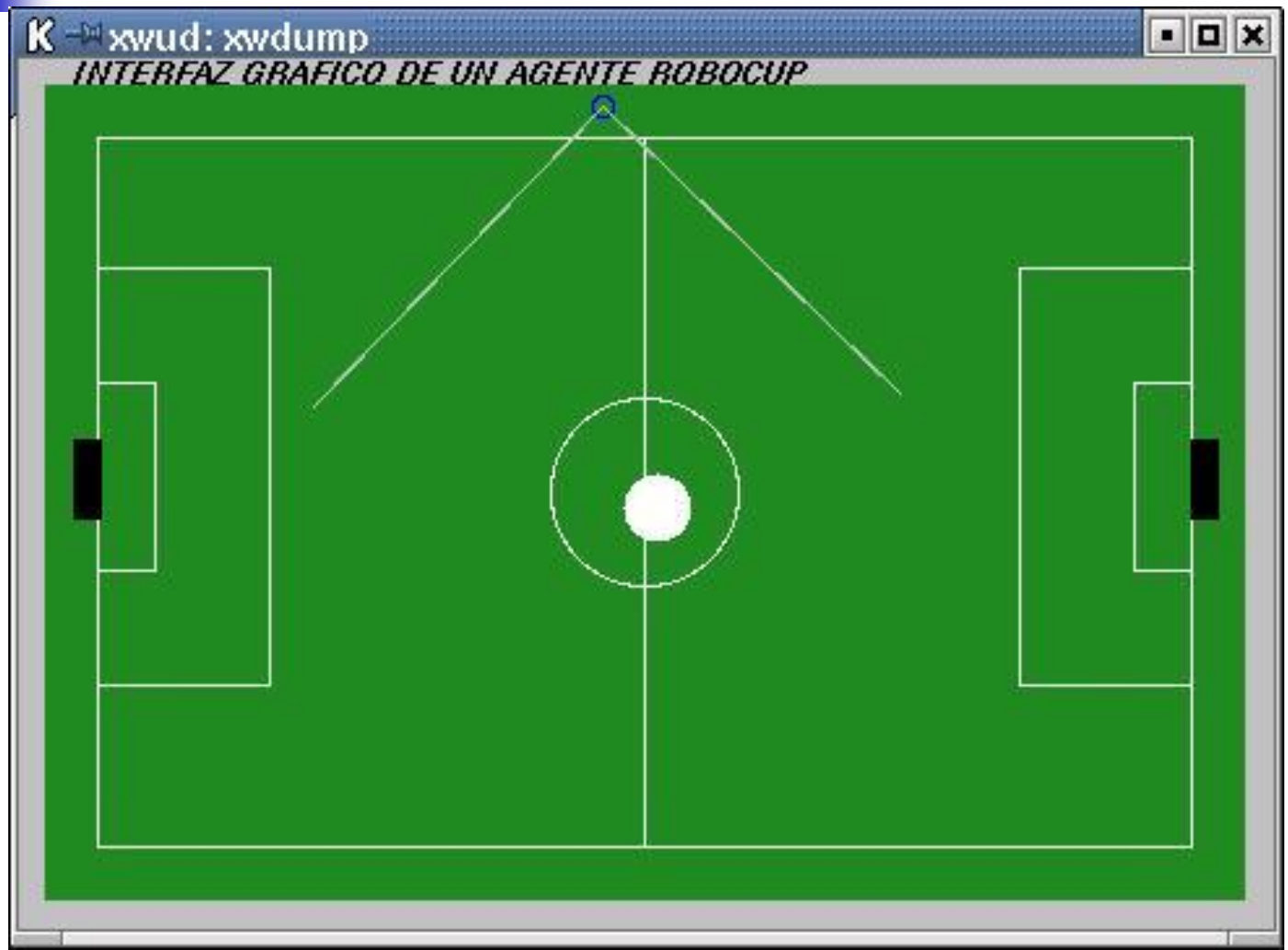
Ejemplo en Robosoccer

Ricardo Aler, Oscar Garcia and Jose M. Valls. 2005. “Correcting and Improving Imitation Models of Humans for Robosoccer Agents”

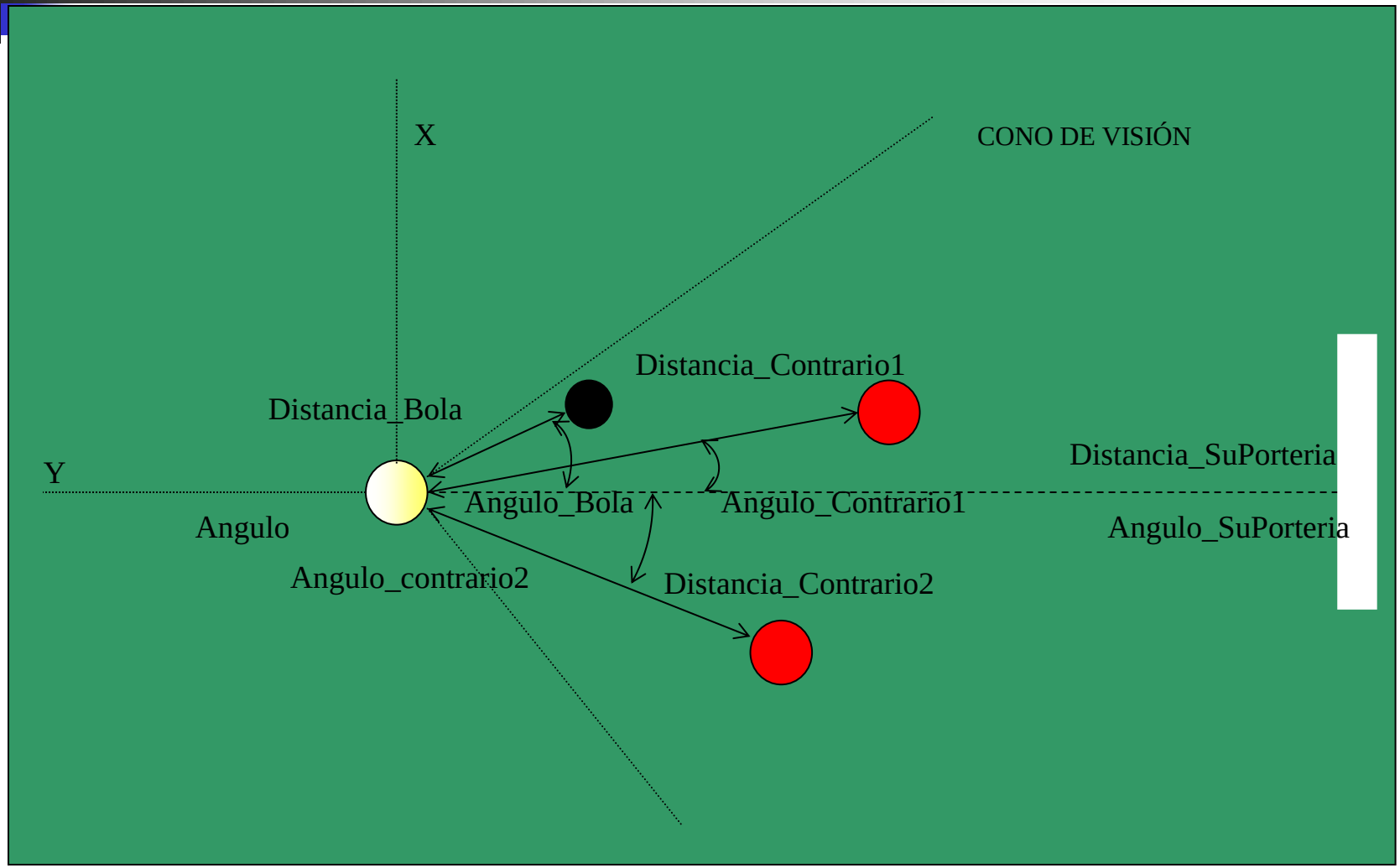
Esquema de aprendizaje



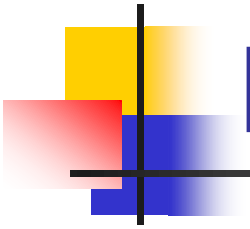
Interfaz para el humano



Entradas del agente en Robosoccer



Entradas del agente en Robosoccer



- Entorno parcialmente observable:
  - Cono de visión
- Visión limitada (según la distancia)
- Adición por parte del simulador de ruido aleatorio en sensores y acciones

Acciones en Robosoccer

Acciones
Avanzar rápido: **dash99**
Avanzar lento: **dash 60**
Girar 10º Derecha: **turn-right-10**
Girar 10º Izquierda: **turn-left-10**
Tirar a puerta: **kick99**
Tiro corto: **kick60**

- Adición por parte del simulador de ruido aleatorio en acciones
- Hay viento, rozamiento, etc
- Turn y dash son incompatibles
- Modelo de Estamina (cansancio)

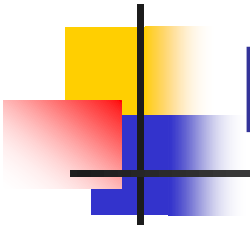
Reglas obtenidas

if ((Angulo_Bola > -37)&&(Distancia_Bola > 1.2) &&(Angulo_Bola <= 24)) {dash99(memoria,puerto); break;}

if ((Angulo_Bola > 19)&&(Angulo_Bola <= 42)&&(X <= 33.9477)) {dash99(memoria,puerto);break;}

if ((Angulo_Bola > 11)) {turn10(memoria,puerto);break;}

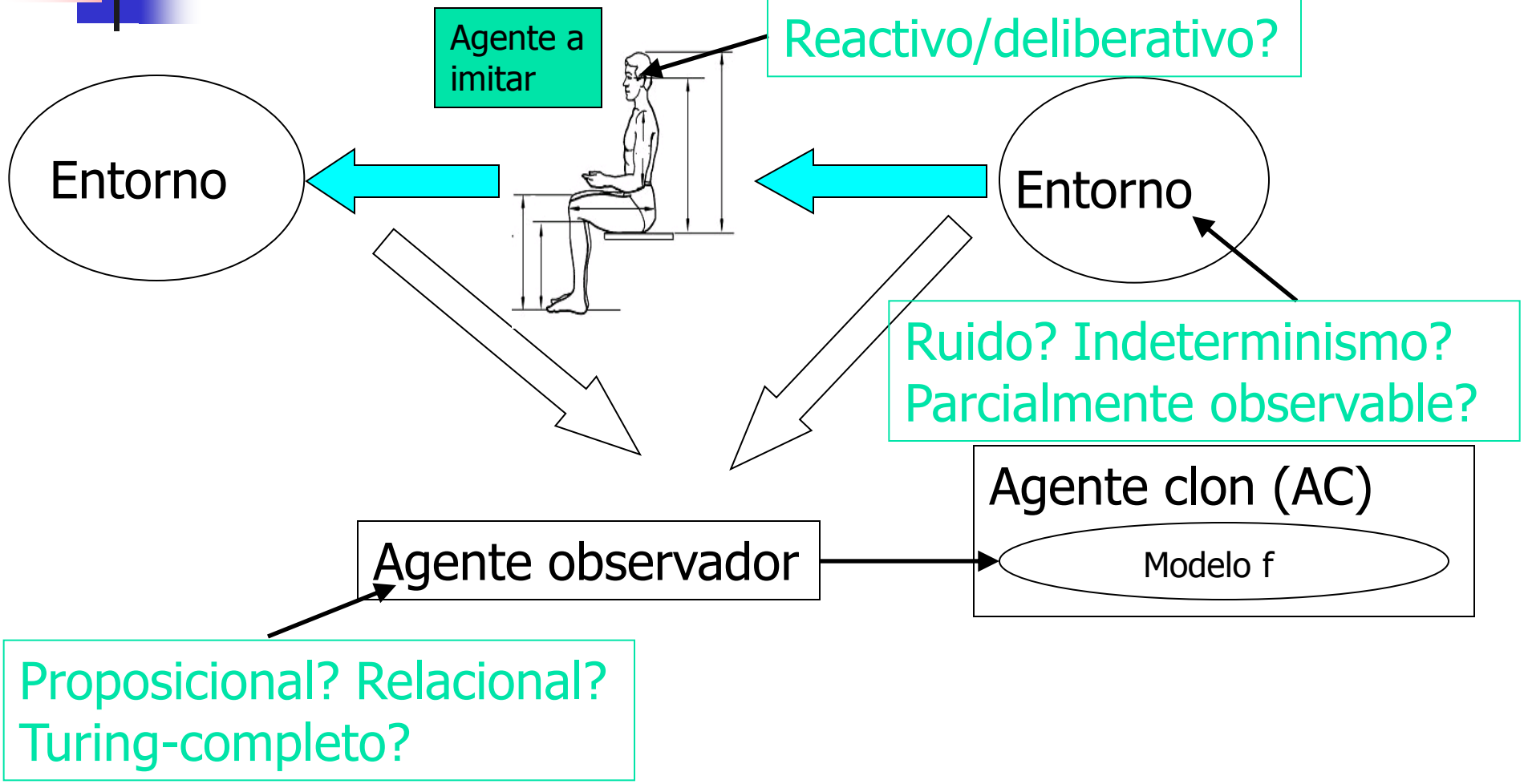
if ((Distancia_Bola <= 0.4)&&(Angulo_Bola <= -20)) {turn10(memoria,puerto);break;}

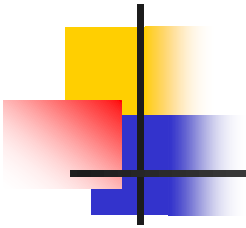


Behavioral Cloning: Utilidad

- Programación de agentes complejos a partir del comportamiento de otro agente ya existente (una especie de ingeniería inversa). Ej: en juegos
- Programación de agentes que imiten el comportamiento humano para juegos
- Transferencia de las habilidades de una persona a un agente. Ej: en juegos
- Se puede usar el modelo para predecir el comportamiento del agente observado (AI) (menos complicado que construir un clon)

BC: Aspectos a considerar

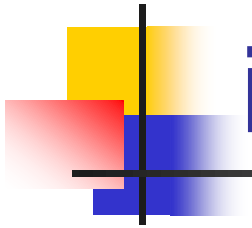




BC: Aspectos a considerar

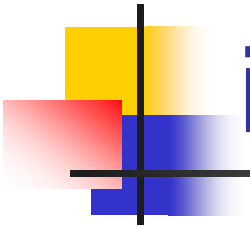
1. Complejidad del agente a imitar: reactivo vs. deliberativo
2. Modelo a aprender: entrada-salida vs. entrada-estadointerno-salida
3. Lenguaje de modelado: proposicional, relacional, programación automática
4. Complejidad del entorno: ruido, nodeterminismo, parcialmente observable, ...

BC: complejidad del agente a imitar



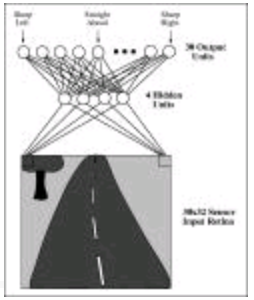
- Tipología de agentes:
  - Reactivos:
    - Sin memoria o estado interno
    - Simplemente reaccionan a la entrada
  - Deliberativos:
    - Tienen memoria / estado interno
    - Deliberan:
      - Hacen planes a largo plazo
      - Tienen objetivos y subobjetivos
      - Predicen del futuro
      - Infieren aquello que no ven
      - Etc.

BC: complejidad del agente a imitar



- Sólo se pueden ver las entradas y salidas del agente a imitar (y a veces esto es también difícil)
- El estado interno no es visible
- Las habilidades (planificación, predicción, inferencia, ...) del agente a imitar no se conocen

BC: Artículos básicos

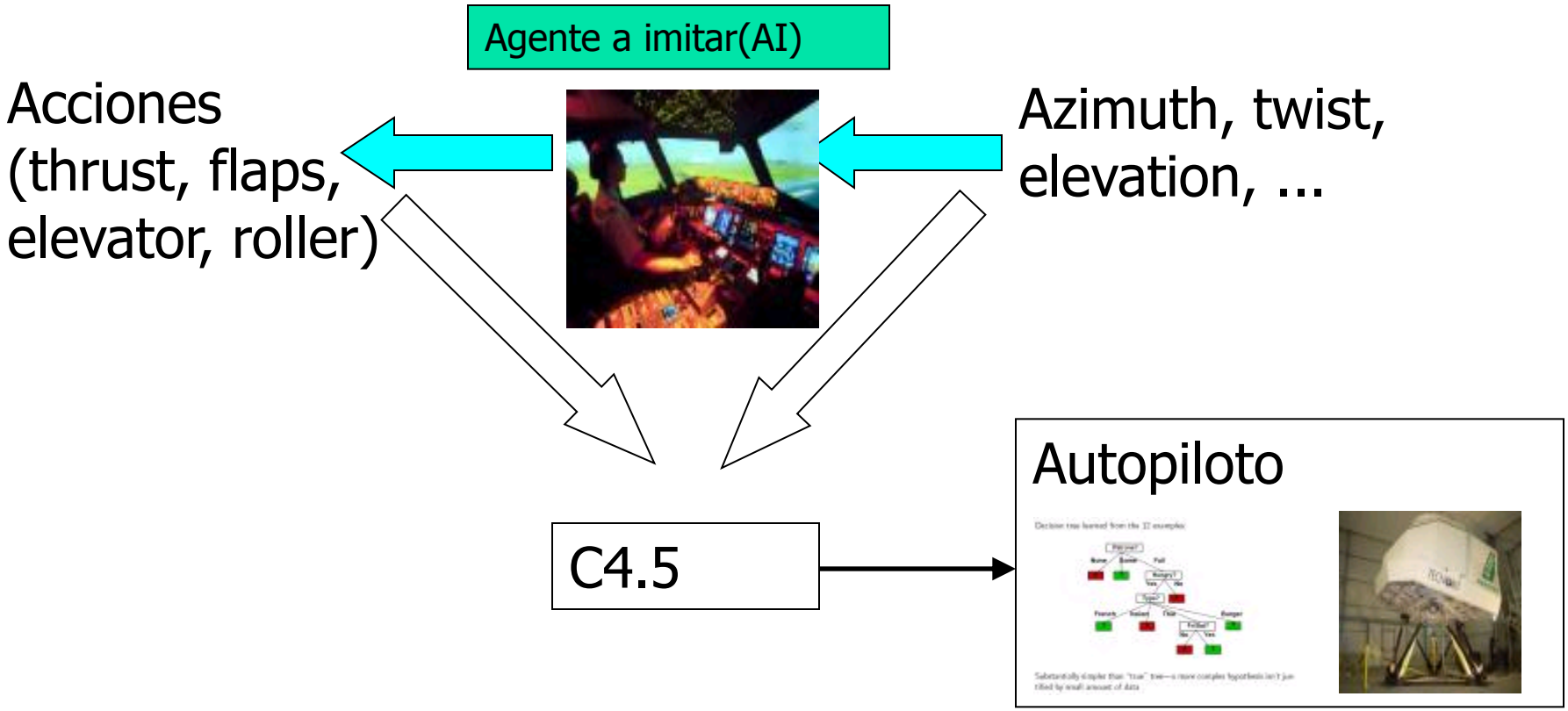


- Pomerleu, 1989. *ALVINN: an Autonomous Land Vehicle in a Neural Network*
- Sammut, Hurst, Kedzierand, Michie. 1992. *Learning to Fly*
- Urbancic, Bratko, 1994. *Reconstructing Human Skill with Machine Learning*
- **KNOMIC**: Van Lent, Laird, 1999. *Learning Hierarchical Performance Knowledge by Observation*. **AIR COMBAT** & **QUAKE II**
- Sklar, Blair, Pollack, 2001. *Training Intelligent Agents Using Human Internet Data*. **TRON**
- Thurau, Bauckhage, Sagerer. 2004. *Imitation Learning at All Levels of Game-AI*. **QUAKE**
- Suc. *Machine Reconstruction of Human Control Strategies*. 2003, **CONTROLADORES: CRANE, BYCICLE**

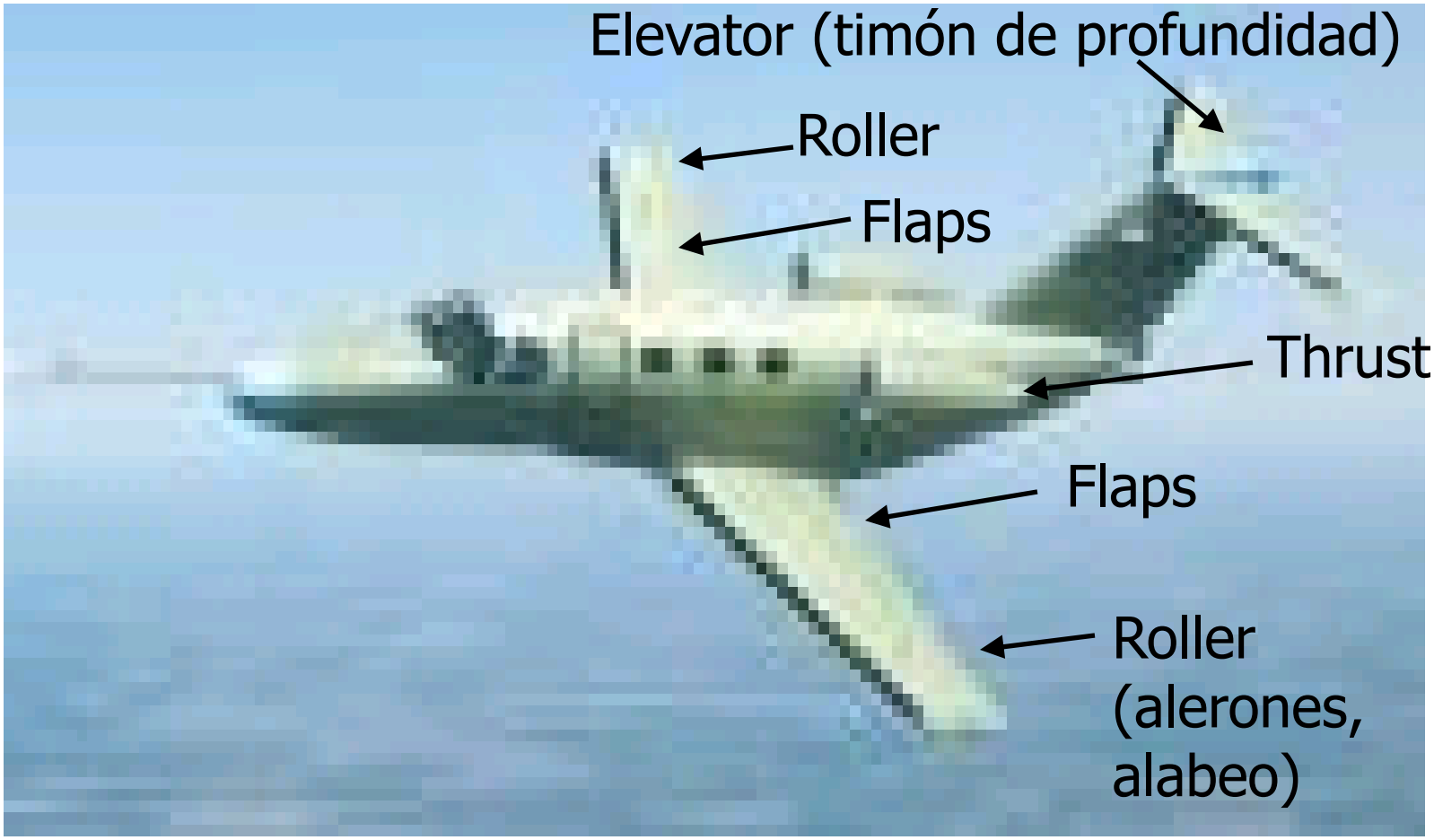
BC: Artículos básicos

- "An example of using imitation on-line, during the actual gameplay, is the XBox game **Forza Motorsport** from Microsoft Game Studios. In this game, the player can train a "drivatar" to play just like himself, and then use this virtual copy of himself to get ranked on tracks he doesn't want to drive himself, or test his skill against other players' drivatars."
  - http://research.microsoft.com/en-us/projects/drivatar/forza.aspx
- Togelius. *Optimization, Imitation and Innovation: Computational Intelligence and Games*. PhD thesis. **CAR RACING**
- Togelius, 2006. *Making Racing Fun Through Player Modeling and Track Evolution.* **CAR RACING**
- Bryant, Miikkulainen, 2007. *Acquiring Visibly Intelligent Behavior with Example-Guided Neuroevolution.* **LEGION**
- Abbott, 2007. Automated Tactics Modeling: Techniques and Applications. Thesis
- General Agent Learning Using First Person Shooter Game Logs by Matthew P. Sheehan. The University of Auckland. http://www.cs.auckland.ac.nz/research/gameai/theses/Matt%20Sheehan%20MSc%20Thesis.pdf
- Bhuman Soni, Philip Hingston. Bots trained to play like a human are more fun. . IJCNN 2008 . 2008

Learning to Fly



Controles básicos del Cessna



Controles del piloto

- Mediante un joystick
- **Elevator**: Orientar el morro arriba/abajo (timón de profundidad, cabeceo/pitch, ángulo de ataque)
- **Thrust**: potencia de los motores (control de gases)
- **Flaps**: frenan e incrementan la sustentación a velocidades bajas
- **Rollers**: control de alabeo (alerones, para girar hacia izquierda o derecha)

Controles del piloto

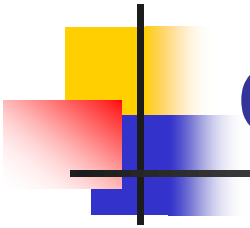
Elevator (Angulo de ataque), control de velocidad

Fig.5.1.1 - Actitudes de morro (vista lateral).

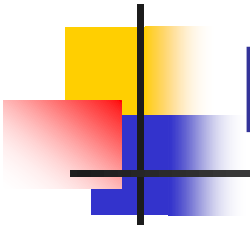
Roller (alabeo), para girar

Fig.5.1.3 - Actitudes de alabeo.

Vuelo: dominio dinámico complejo



- Volar es un problema dinámico complicado
- Aumentar la potencia hace que el avión suba (incluso decelera algo)
- Hacer que el morro vaya hacia arriba hace que la velocidad descienda (por resistencia)
- Es muy difícil controlar altura y velocidad de manera independiente (a diferencia de un coche)
- Ajustes continuos de ambos mandos (thrust y elevator)
- La reacción del avión a los controles también depende de su propia estructura
- Es difícil programar controladores en situaciones de viento fuerte
- Es un sistema **dinámico** (continuo) y **no-determinista**



Learning to Fly

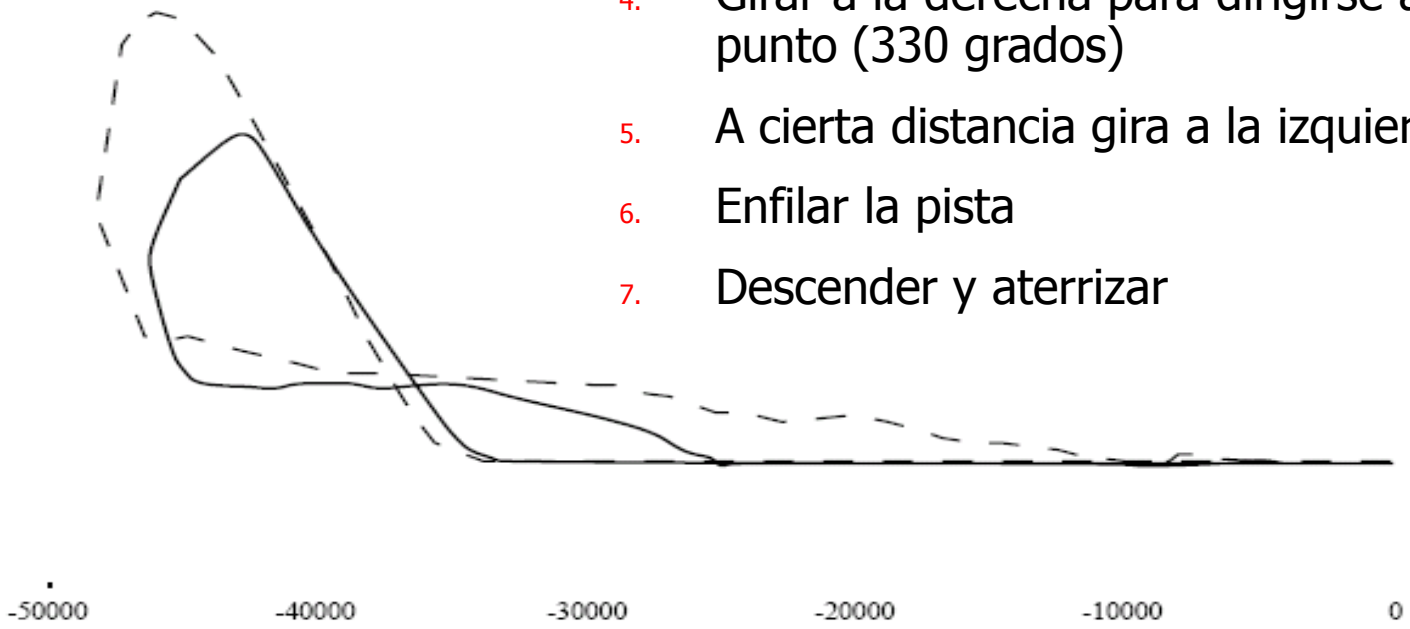
- Humano en un simulador
- Se graban sus reacciones frente a situaciones (fichero de log)
- Se utiliza el log para construir un árbol de decisión
- Se prueba el árbol de decisión en el simulador en modo autopiloto
- Hipótesis de partida: es mejor partir de un piloto experto que generar el controlador de manera aleatoria

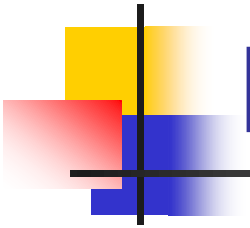
Configuración experimental

- Se eligió una avioneta Cessna
- Se utilizaron 3 pilotos
- En cada vuelo se guardan 1000 sucesos
- Con 30 vuelos por piloto, 3 pilotos = 90000 eventos

Plan de vuelo

1. Comienzo orientado al norte (0 grados)
2. Despegar y subir a 2000 pies
3. Estabilizar y volar 32000 pies
4. Girar a la derecha para dirigirse a determindo punto (330 grados)
5. A cierta distancia gira a la izquierda hacia la pista
6. Enfilar la pista
7. Descender y aterrizar





Datos guardados del suceso

- Cada vez que el piloto usa un control, se guarda un suceso en el log
- Elevation: elevación del “morro” del avión
- Azimuth: orientación del avión (brújula)
- Twist: ángulo de las alas con la horizontal
- Thrusts: potencia de gases

Datos guardados del suceso

on_ground	boolean: is the plane on the ground?
g_limit	boolean: have we exceeded the plane's g limit
wing_stall	boolean: has the plane stalled?
twist	integer: 0 to 360° (in tenths of a degree, see below)
elevation	integer: 0 to 360° (in tenths of a degree, see below)
azimuth	integer: 0 to 360° (in tenths of a degree, see below)
roll_speed	integer: 0 to 360° (in tenths of a degree per second)
elevation_speed	integer: 0 to 360° (in tenths of a degree per second)
azimuth_speed	integer: 0 to 360° (in tenths of a degree per second)

Datos guardados del suceso

airspeed	integer: (in knots)
climbspeed	integer: (feet per second)
E/W distance	real: E/W distance from centre of runway (in feet)
altitude	real: (in feet)
N/S distance	real: N/S distance from northern end of runway (in feet)
fuel	integer: (in pounds)
rollers	real: ±4.3
elevator	real: ±3.0
rudder	real: not used
thrust	integer: 0 to 100%
flaps	integer: 0°, 10° or 20°

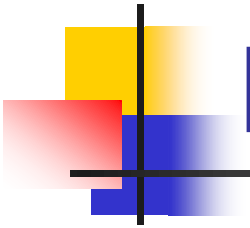
Datos guardados del suceso. Limitaciones.

- El piloto para maniobrar, se fija en un punto del horizonte (ej: comienzo de la pista). Pero este punto es desconocido para el sistema y no se guarda en el log
- Este es un ejemplo de cómo no siempre todos los datos relevantes están disponibles para ser almacenados
- En este caso, se espera que el resto de datos almacenados suplan los datos ocultos

Datos guardados del suceso. Limitaciones.

- Se debería guardar las parejas *(estímulo, respuesta)*, donde *respuesta* es la reacción frente al *estímulo*
- Pero siempre hay un retardo desconocido entre *estímulo* y *respuesta*
- Solución parcial: se determina el retardo aproximado para cada piloto (entre 1 y 3 segundos)

Datos guardados del suceso. Limitaciones.



- Incluso aunque en este problema el plan de vuelo está bien especificado, puede haber grandes variaciones en la respuesta del piloto: ante la misma situación, distinta respuesta
- Esto es incluso peor si se utilizan distintos pilotos para generar los logs
- Además, al haber ruido en el dominio, es difícil para el algoritmo de aprendizaje distinguir entre ruido y variación
- Solución parcial: aprender un árbol de decisión para cada piloto y utilizar muchos vuelos

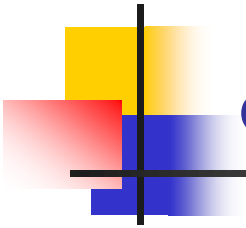
Ejemplo de variabilidad en el vuelo

- Para mantener un vuelo estable (horizontal):
  1. Aumentar o disminuir la potencia (correcto)
  2. Aumentar o disminuir la elevación (menos correcto)
- Dos pilotos utilizaron 2. y uno 1.

Disminución de la ambigüedad

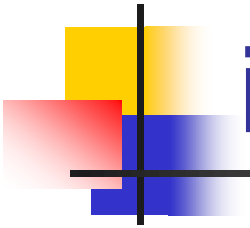
- En diferentes etapas del vuelo, el piloto puede reaccionar de manera distinta ante situaciones parecidas
- Para disminuir la ambigüedad, dividir el vuelo en **7 etapas** (despegue, vuelo recto, giro, ..., aterrizaje)
- Puesto que el piloto sabe que hay que cumplir 7 subobjetivos, se le da al sistema de aprendizaje la misma de información
- Lo ideal sería que el propio sistema de aprendizaje descompusiera la tarea global en los 7 subobjetivos

Árboles de decisión aprendidos



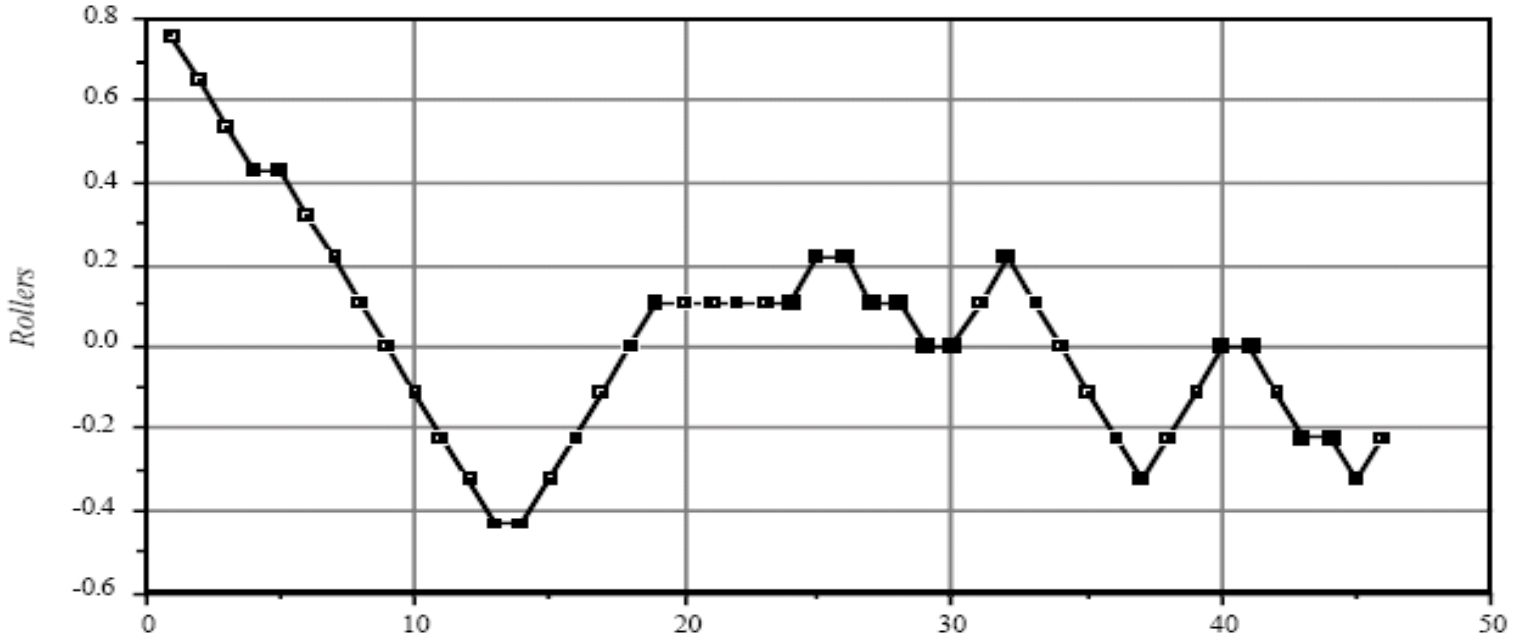
- Se utiliza C4.5
- Para cada control, se aprende un árbol de decisión diferente (es necesario discretizar la clase para C4.5)
  - Elevator
  - Roller
  - Thrust
  - Flaps
- Además, si queremos aprender un árbol de decisión para el elevator, el estado del roller, thrust y flaps se utilizan como atributos

Controles absolutos e incrementales



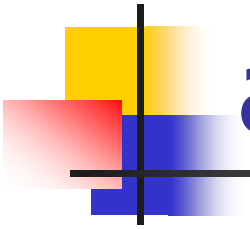
- Thrust y Flaps se pueden activar y dejarlos así hasta que se alcanza el estado deseado (ej: altitud deseada)
- Rollers y Elevator requieren ajustes continuos. Ej: mover el joistick hacia la izquierda, centrarlo, girarlo otra vez, ... De otra manera el avión se daría la vuelta.
- En suma, Thrust y Flaps no requieren un ajuste constante, pero Rollers y Elevator si.

Ejemplo de uso de Rollers



Sólo se guardan los cambios de tendencia. Después cuando se usen, habrá que interpolar.

Utilización de las reglas aprendidas



- Utilizar poda de C4.5 para eliminar las inconsistencias en el manejo de los controles del piloto (clean-up effect)
- Las reglas C4.5 se convierten a reglas C:
  - *if (condicion) {accion}*
- Es necesario introducir el retardo (la condición se refiere a un tiempo anterior a la acción)
- Es necesario introducir la gradualidad de cambio en los controles (ej: los Rollers no se fijan instantaneamente)

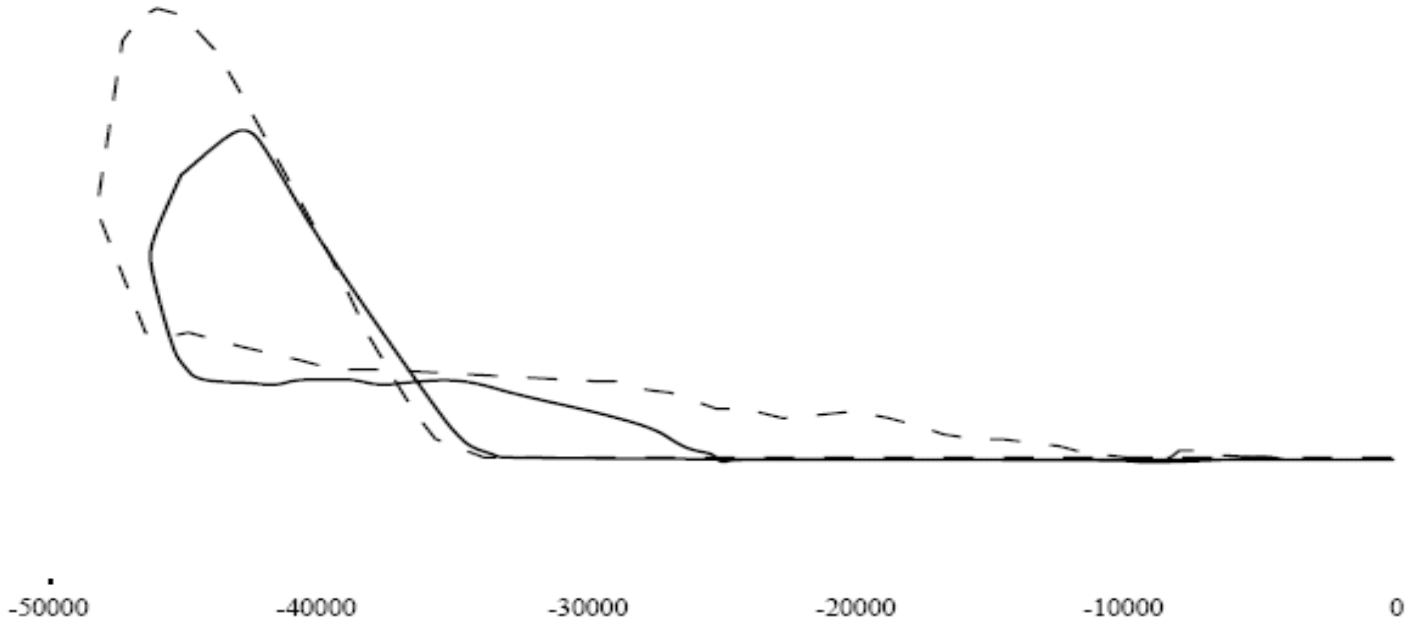
Reglas etapa 1 (Despegue)

```
elevation > 4 : level_pitch
elevation <= 4 :
| airspeed <= 0 : level_pitch
| airspeed > 0 : pitch_up_5
```

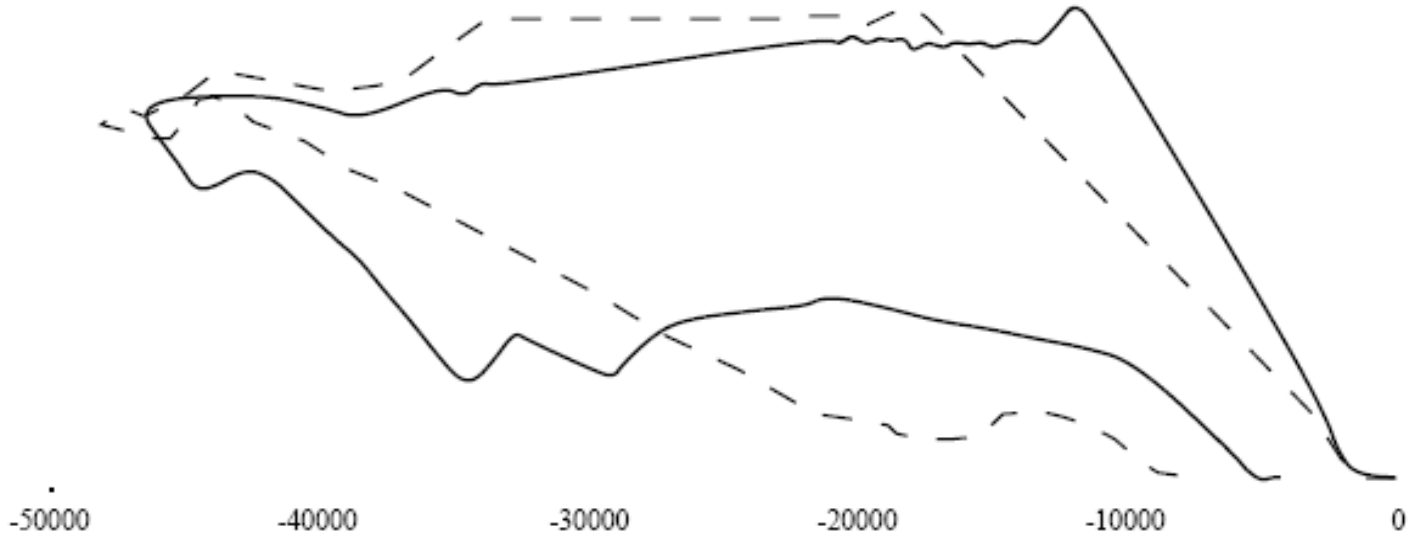
Reglas etapa 3 (vuelo estable)

```
climbspeed <= 13 : level_pitch
climbspeed > 13 : pitch_down_1
```

Vuelo real y vuelo controlado por reglas

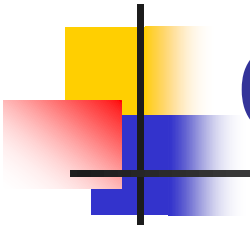


Vuelo real y vuelo controlado por reglas (altitud)



Conclusiones. Detalles a tener en cuenta en otros dominios

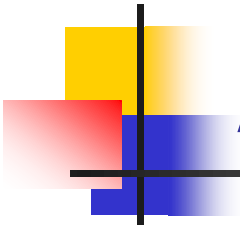
- Puede existir un retardo entre situación y reacción a la situación
- El piloto humano puede anticiparse a lo que va a pasar y comenzar la maniobra correctora en avance (la reacción no se basa solo en la situación, sino también en una predicción):
  - El modelo es reactivo, pero el agente es deliberativo
- El piloto humano puede hacer muchas más correcciones de las necesarias para conseguir un estado deseable. Por medio de la poda, se puede "limpiar" la traza y conseguir un clon de ejecución más suave (clean-up effect)



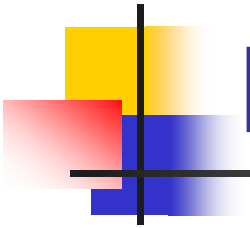
Conclusiones. Ambiguedad

- A pesar de que la tarea está muy constreñida, existe mucha ambigüedad para las reglas
- Reglas: IF situación THEN acción
- Si en la traza ocurre que:
  - Situación(velocidad=3,t1), acción subir
  - Situación(velocidad=3,t2), acción estable
- Entonces hay dos situaciones similares en tiempos distintos que requieren acciones distintas
- Pero, ¿en base a qué se toman decisiones distintas si la situación es la misma?

Conclusiones. Fuentes de Ambiguedad

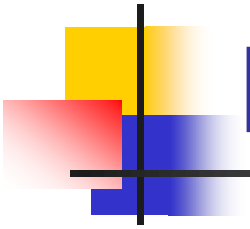


- Distintos pilotos pueden reaccionar de distinta manera ante situaciones similares
- Esto puede ocurrir incluso con el mismo piloto (esto no siempre es corregible con la poda)
- Y también en distintas etapas del vuelo
- E incluso dentro de las etapas
- No siempre todos los datos que usa el piloto pueden ser utilizados por las reglas (ej: punto en el horizonte hacia el que dirigirse, memoria del piloto acerca de que una zona es de turbulencias, ...)



Extensiones

- Lo ideal es que el sistema, a partir de los logs de varios pilotos, identificara automáticamente las inconsistencias:
  - Que ciertos datos corresponden a distintos estilos de vuelo (distintos pilotos o incluso para el mismo piloto)
  - Que existen distintas etapas
  - …



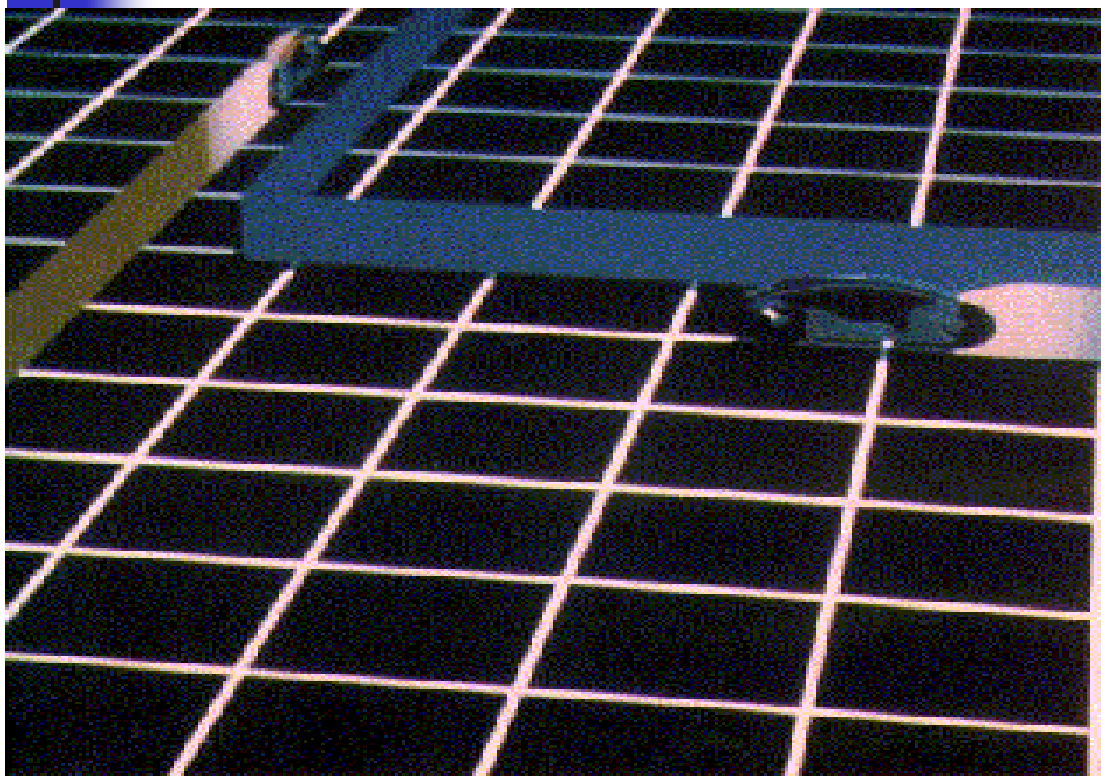
Extensiones para robustez

- Reacción a fenómenos puntuales (ej: turbulencias) que sitúan al avión fuera de la situación estable (ej: volando estable y recto)
- Para cada etapa, es preferible aprender por separado:
  - Reglas de mantenimiento de la situación estable
  - Reglas de consecución de la situación (goal-seeking) cuando ocurren fenómenos que llevan al avión fuera de su estado estable

Extensiones para flexibilidad

- El nivel de descripción de las acciones es de demasiado bajo nivel (thrust, flaps, elevator, roller)
- El piloto piensa en términos de más alto nivel: conseguir vuelo recto y estable, giro suave, giro brusco, etc.
- Es decir, cada etapa consiste a su vez de sub-objetivos.
- Sería conveniente definir a priori los subjetivos y que el sistema aprendiera reglas de la forma:
  - SI situacion=x ENTONCES subobjetivo y
  - SI situación=x Y subobjetivo=y ENTONCES z

BC aplicado a TRON [Sklar et al. 99]



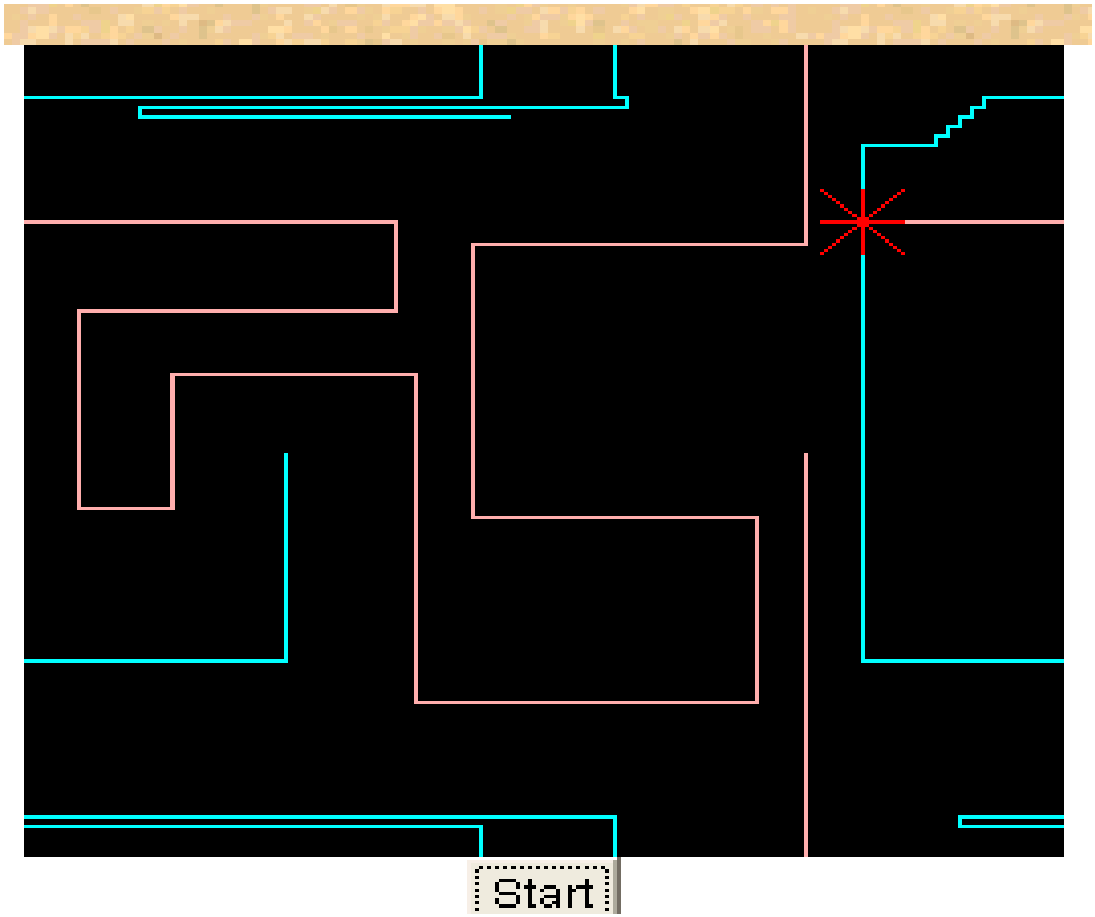
Acciones:

- Left
- Right
- Straight

Objetivo: generar una población de agentes de competencia variada para enfrentarlos a humanos

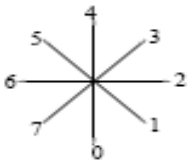
http://demo.cs.brandeis.edu/tron/

TRON. El juego

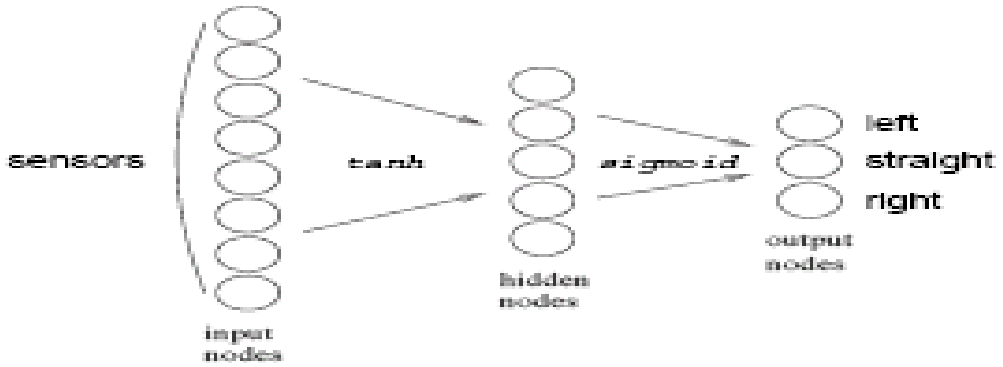


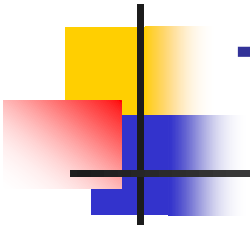
TRON

- 8 Sensores: indican distancia al objeto más cercano



- Aprendizaje: ANN, Hinton crossentropy (se aprende la probabilidad de la acción, bueno contra inconsistencias en los humanos)

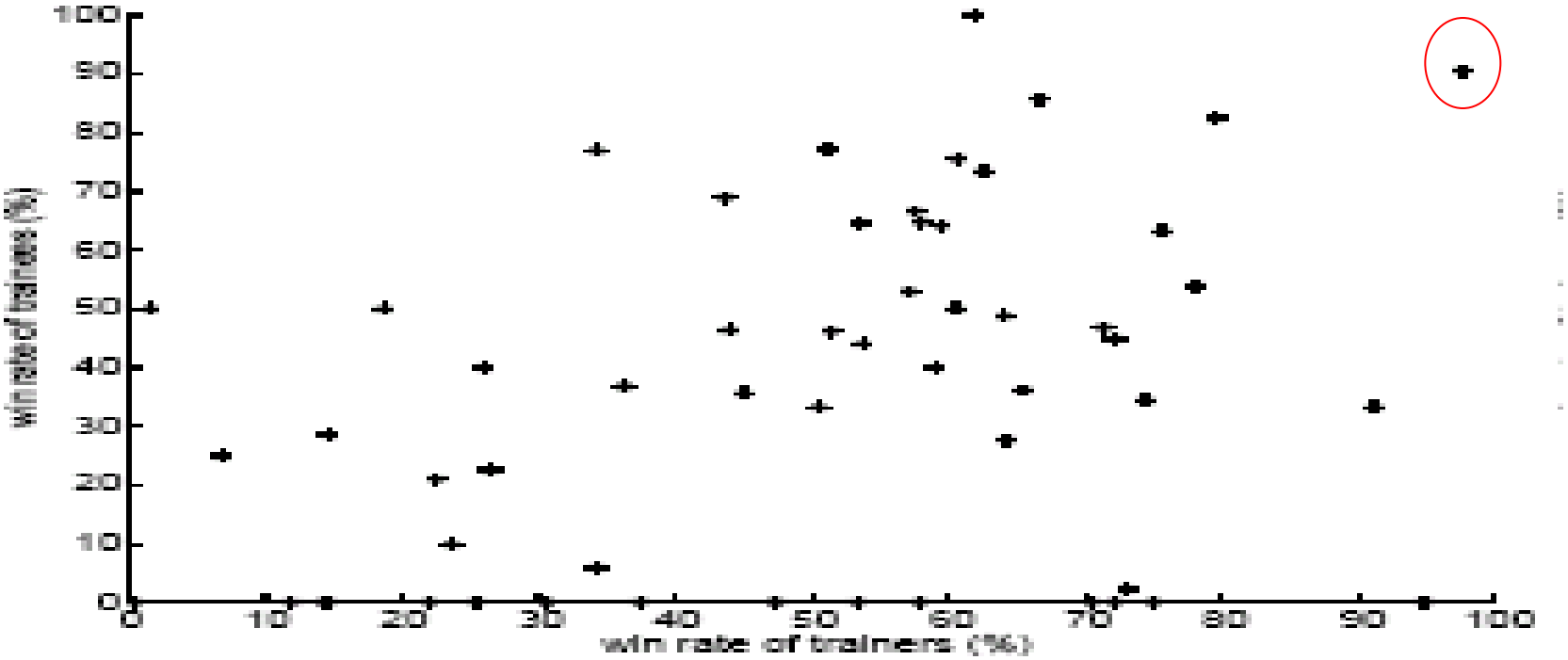


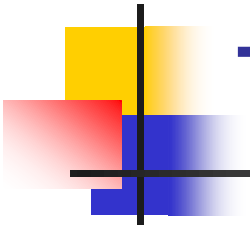


TRON

- Mayor dificultad: el 90% del tiempo, se avanza (*straight*)
- Esto hace que sea difícil aprender *left* y *right*
- Pero si se gira mas de la cuenta, se muere antes
- Aprendizaje con coste:
  - Enfatizar las clases minoritarias (*left* y *right*)
  - Tener cuidado de que la frecuencia de giros de la red de neuronas sea similar a la del humano

TRON. Resultados

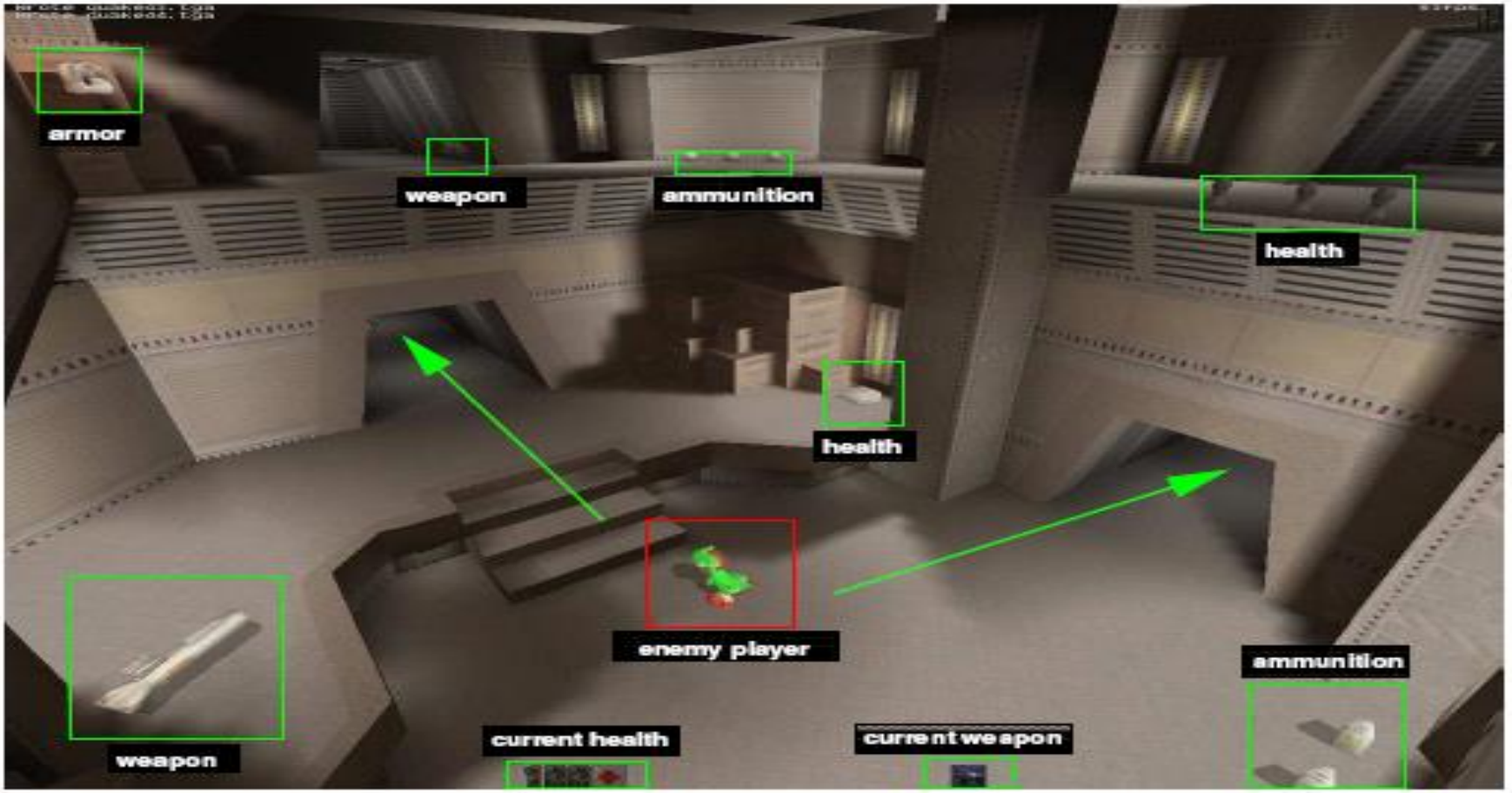




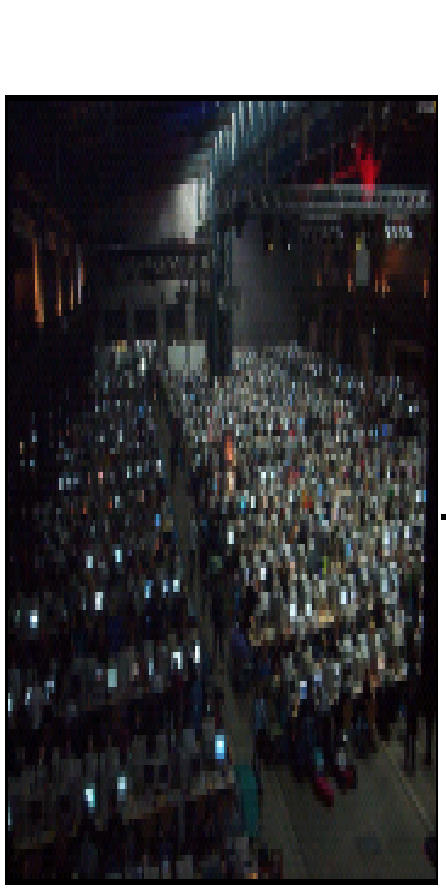
TRON. Conclusiones

- Se pueden aprender buenos jugadores de TRON con BC
- No está claro que imiten a los humanos:
  - La correlación entre buenos jugadores y buenos clones es alta
  - La correlación entre movimientos del humano y el clon es más baja
- El problema de las inconsistencias solventado con la cross-entropía
- Dominio muy restringido

Quake II. [Thurau et al]



Quake II. Niveles de BC (estrategia, táctica, reactivo)



Training Samples

Behavior Learning

Strategy

item pickups, long term goals ...

Tactics

anticipation, smart localized behaviors

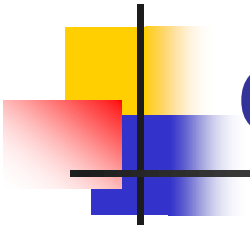
Reactive Behaviors

basic movements, aiming ...

Motion Modeling

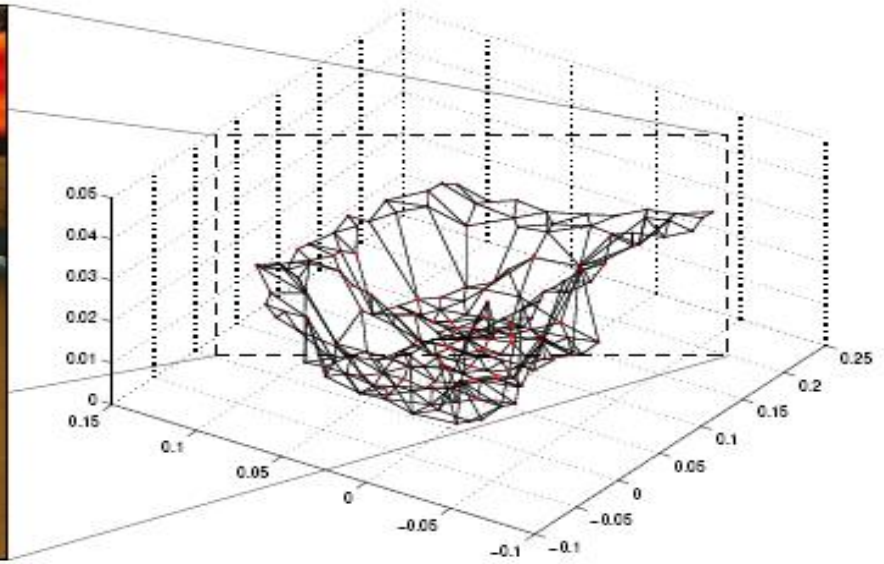
Game Agent

Quake II. Aprendizaje de estrategias



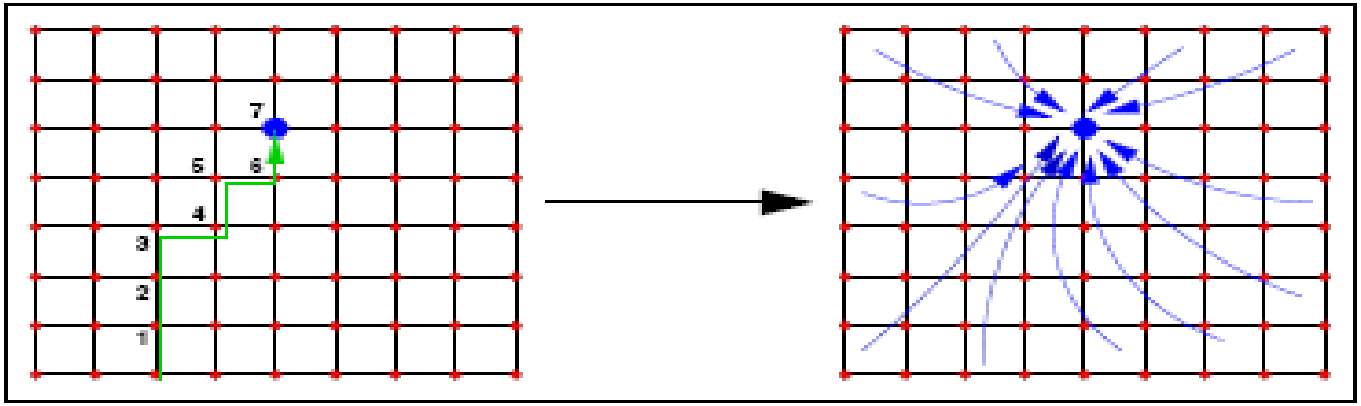
- Estrategia: a qué **lugar** del mapa dirigirse a continuación
- Depende de la situación del jugador:
  - Si está herido, dirigirse al medicaid
  - Si no, ir a buscar arma potente
  - O controlar cierta zona de paso de jugadores contrarios
  - ...

Quake II. Mapas topológicos



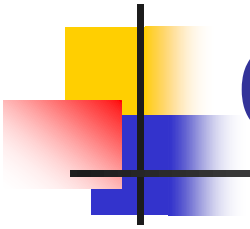
Se construye una colección de puntos visitados por los jugadores

Quake II. Estrategias. Campos de fuerza (o potencial)



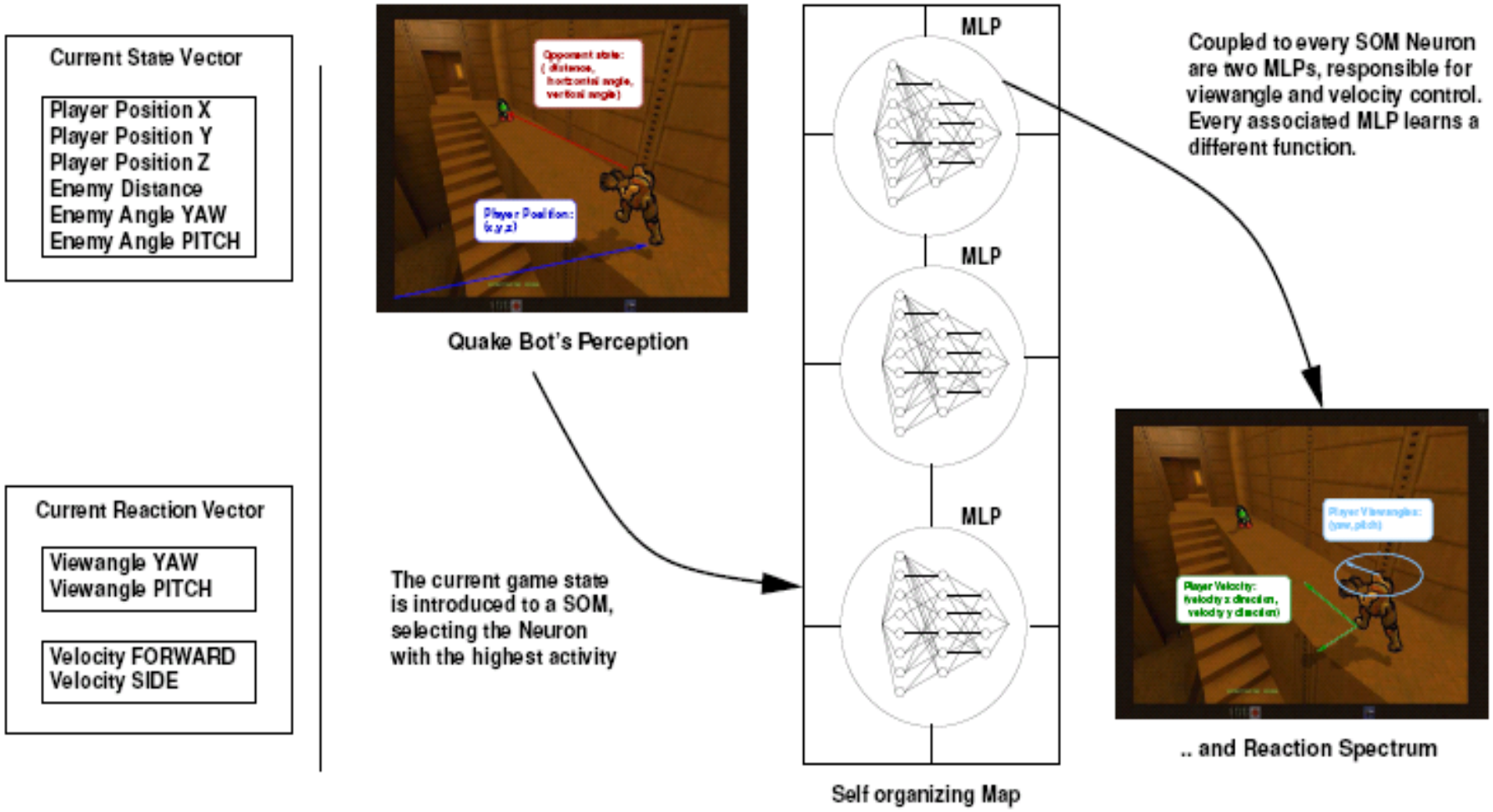
- Para evitar quedarse parado en zonas de fuerza baja, dejar caer feromonas temporales para no pasar por el mismo sitio
- Se aprenden campos de fuerza distintos para situaciones distintas (armas, armadura, salud)
- Las situaciones típicas se obtienen mediante "neural gas" (parecido a clustering con k-medias)

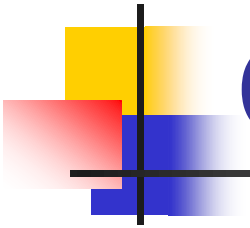
Quake II. Aprendizaje reactivo



- Estado de juego = (X,Y,Z,coords enemigo, ...)
- Primero se generan situaciones de juego típìcas (prototipos) mediante clustering de los estados de juego (SOM = self-organizing-maps). Se podría hacer con k-medias
- Después se aprende una ANN para cada cluster

Quake II. Aprendizaje reactivo

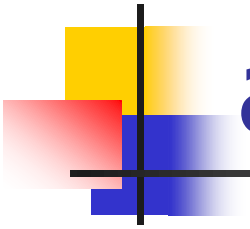




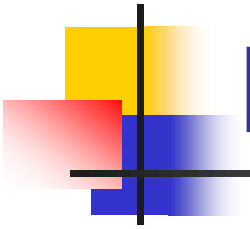
Quake II. Otros

- Aprendizaje de tácticas (qué armas usar en cada momento)
- Aprendizaje de movimientos (para que sean suaves como los de los humanos)

BC in the Robosoccer. [Aler et al.2005]



- Aler, Valls, García. "Correcting and Improving Imitation Models of Humans for Robosoccer Agents". CEC'05
- http://tracer.uc3m.es/problems/robosoccer/robosoccer.html
- Ejemplo de uso de BC como punto de partida para otros métodos (evolutivo en este caso), especialmente si BC es complicado en el dominio



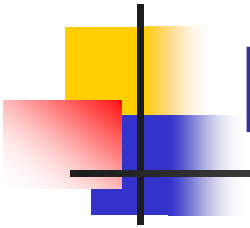
BC in the Robosoccer

- Etapas:

1. Generar reglas iniciales con BC
2. Mejorar esas reglas con un algoritmo evolutivo, “sembrando” la población inicial

Robosoccer

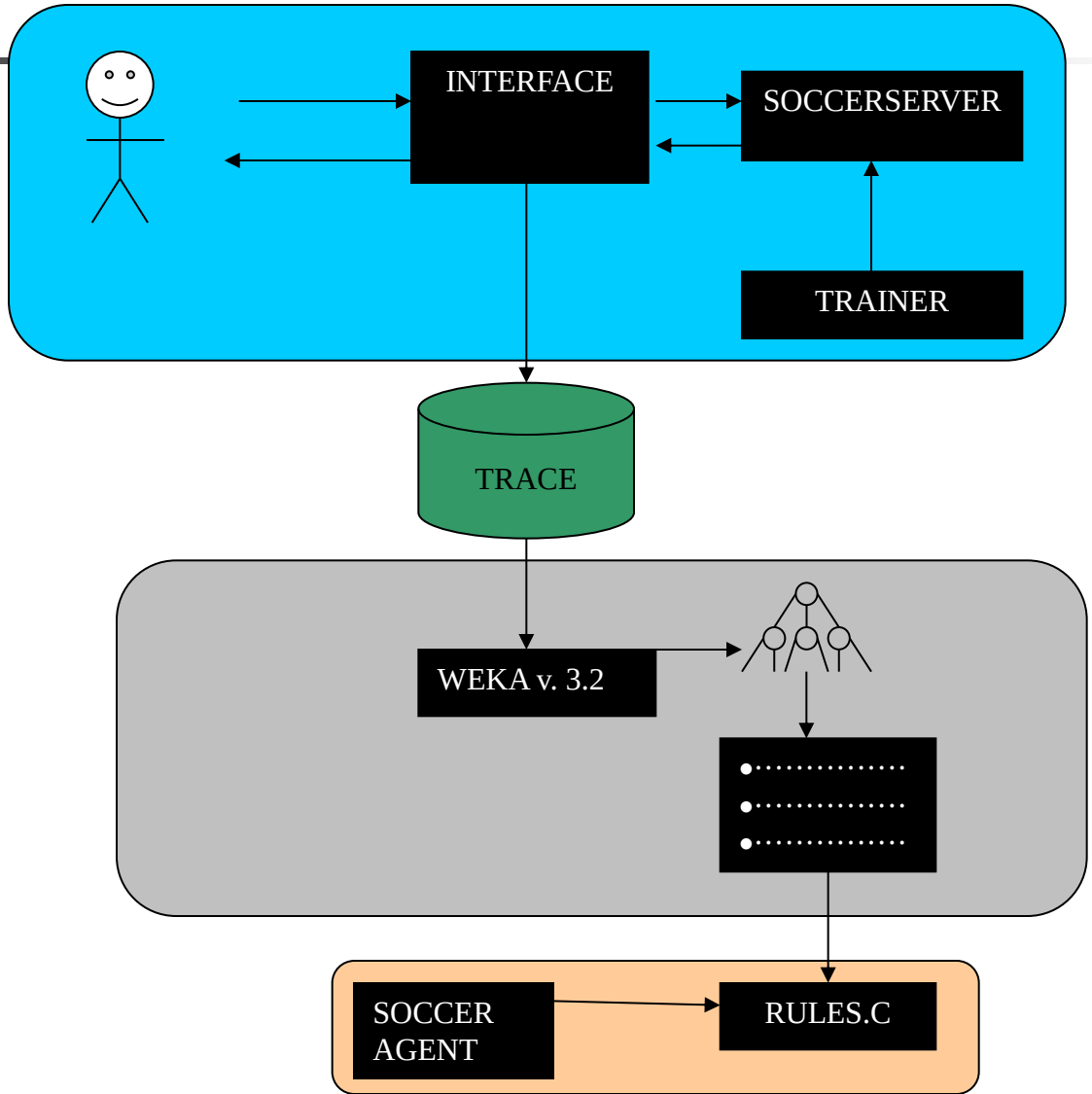

xvud: soccermonitor
Kick Off
Quit
agente: 0
kick_off_l
41
Team_R
(referee kick_off_l)
Recv agente_1: (dash 99)
Recv agente_1: (dash 99)
Recv agente_1: (dash 99)
Recv agente_1: (dash 99)
Recv agente_1: (dash 99)



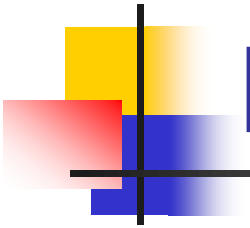
BC in the Robosoccer

- Suposiciones / decisiones de diseño:
  - El humano es un agente reactivo -> el humano tuvo que ser tan consistente como fuera posible
  - El lenguaje de modelado PART es proposicional (atributo-valor). Se aprenden conjuntos de reglas

Esquema de BC



Behavioral Cloning en Robosoccer

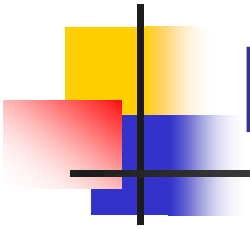


- Se construyó un interfaz para el humano
- Tarea (táctica): con un delantero, regatear a 3 defensas y golear a un portero
- Equipo "no campeón", pero tarea difícil:
  - Son mas y juegan cooperativamente
  - Tienen un portero
  - Se pueden turnar (nuestro jugador se cansa)

Ejemplos de reglas aprendidas

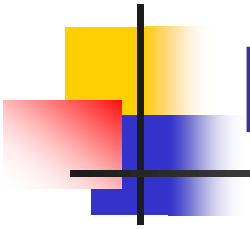
- **if** ((Distance_Ball <= 0.1) &&
  (Visible_Goal == 1))
  {kick99}
- **elsif** ((Visible_Ball == 0))
  {turn10}
- **else** {dash99}

Behavioral Cloning en Robosoccer. Resultados

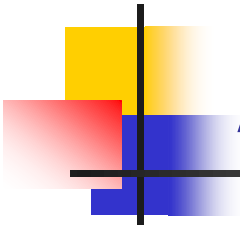


- 24594 (e,s) -> 164 reglas (93%):
  - 5-4, 2-4, 3-4, 4-5, 2-4, 3-4
  - 1 partido ganado, 5 perdidos
  - +19 goles vs -25 goles
- 164 reglas poda -> 69 reglas (92%):
  - 3-4, 2-4, 3-3, 2-5, 3-1, 4-3
  - 2 partidos ganados, 1 empate, 3 perdidos
  - +17 goles vs. -20 goles

Conclusiones BC en Robosoccer



- Se pueden aprender reglas efectivas que marcan goles, pero se puede mejorar
- El humano encontró difícil batir al contrario
- El humano tuvo que comportarse de la manera más reactiva posible
- En lugar de mejorar BC, se mejorarán las reglas mediante una técnica evolutiva



Algoritmo evolutivo

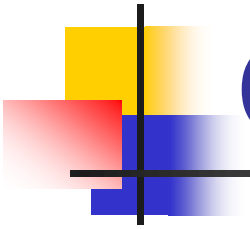
CREATE POP with M individuals
EVALUATE POP
LOOP UNTIL termination criterion
  FOREACH $IND \in POP$
    $OFFSPRING = \emptyset$
    REPEAT N times
      $OFFSPRING \leftarrow$ MUTATE(IND)
    EVALUATE $OFFSPRING$
    $POP \leftarrow$ BEST M individuals from $OFFSPRING$

Ejemplo de individuo

```
IF (angle-ball > -37) AND
   (distance-ball > 1.2) AND
   (angle-ball <= 24)
THEN dash99
ELSE IF
   (angle-ball > 19) AND
   (angle-ball <= 42)
THEN dash99
ELSE turn10
```

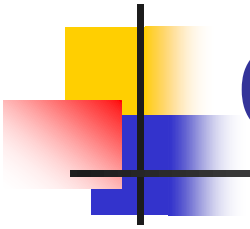
Gramática para modificar individuos

```
RULE          :   CONDITIONS ACTION
CONDITIONS    :   CONDITION CONDITIONS
CONDITION     :   ATTRIBUTE OPERATOR VALUE| AT_BOOL OP_BOOL
   VAL_BOOL
ATTRIBUTE     :   Distance_Ball | Angle_Ball | X | Y| Angle |
   Angle_Goal | Distance_Goal| Distance_Opponent1|
   Distance_Opponent2| Angle_Opponent1| Angle_Opponent2
OPERATOR      :   >|<|>=|<=|==|!=
VALUE         :   REAL
AT_BOOL       :   Visible_Ball | Visible_Goal |
   Visisble_Opponent1| Visible_Opponent2
OP_BOOL       :   == | !=
VAL_BOOL      :   0|1
ACTION        :   dash99| dash60| turn10| turnminos10| kick60|
   kick99
```



Operadores genéticos

- Inserta/borra/modifica reglas completas
- Inserta/borra/modifica condiciones
- Ajuste fino/grueso de valores numéricos
- Modifica acciones (parte derecha) de la regla



Computación de fitness

- Jugar contra **CMUnited99** (ganador de Robocup99)
- Función de fitness:
  1. Maximiza goles
  2. Minimiza tiempo para marcar
  3. Minimiza distancia final agente-portería
  4. Minimiza distancia final agente-pelota
  5. Minimiza número de reglas

La función de fitness

- G: Number of goals scored
- T: Time to first goal scored
- D_g: Distance of the agent to the opponent goal at the end of the game
- D_b: Distance to the ball at the end of the game
- N_r: Number of rules

Fitness = jugar varios partidos y calcular la media

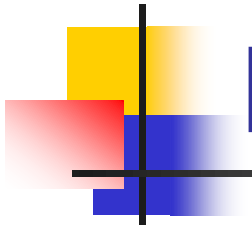
$$K_1 G + K_2 \frac{T_{\max} - T}{T_{\max}} + K_3 \frac{30 - N_r}{30} + K_4 \frac{(52 - D_g)}{52} + K_5 \frac{52 D_b}{52}$$

(1)

Secuencia de Experimentos

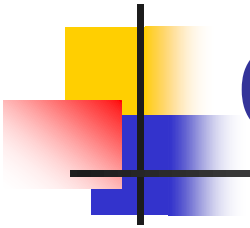
- **Exp. 1:** 1 portero
- **Exp. 2:** 1 portero, 1 defensa estático
- **Exp. 3:** 1 portero, 1 defensa estático
- **Exp. 4:** 1 portero, 1 defensa

- **Test:** 1 portero, 1 defensa. 30 casos de prueba



Resultados

Exp.	Goles marcados	Número de reglas
0	0/30	69
1	1/30	69
2	2/30	70
3	9/30	72
4	17/30	31



Conclusiones

- Behavioral cloning genera modelos razonables en el simulador de Robosoccer
- Un algoritmo evolutivo puede adaptar y mejorar las reglas