Paradigmas en Robótica

Fundamentos de Robótica Autónoma

MINA - Facultad de Ingeniería - Udelar

Contenido

1. Contexto
2. Casos de estudio para:
    a. Jerárquico
    b. Reactivo
    c. Híbrido

Paradigmas

Los paradigmas se describen de dos maneras:

- Por la relación entre las primitivas SENSAR, PLANIFICAR y ACTUAR.
- Por la manera en que los datos sensoriales son procesados y distribuidos en el sistema.

Primitiva robótica	Entrada	Salida
Sensar (SENSE)	Datos de los sensores	Información sensada
Planificar (PLAN)	Información (sensorial o cognitiva)	Directivas
Actuar (ACT)	Información sensada o directivas	Comandos a los actuadores

Arquitecturas

Las arquitecturas proporcionan la manera general de organizar un sistema de control.

Describe un conjunto de componentes y la forma en que interactúan.

Las características más importantes a tener en cuenta al momento de evaluar una arquitectura son:

- Modularidad
- Lugar de aplicación
- Portabilidad
- Robustez

Caso de estudio: STRIPS

Shakey (1967-1971)

- “Primer robot con IA”
- Desarrollado por el Stanford Research Institute (SRI) para DARPA 1967-9.
- Usa STRIPS para determinar qué acción tomar.

> https://www.youtube.com/watch?v=GmU7SimFkpU

Shakey + STRIPS

El robot Shakey necesitaba un algoritmo de propósito general para planificar de forma de alcanzar su objetivo.

- El método utilizado es una versión simplificada del General Problem Solver (GPS), denominado STRIPS (Stanford Research Institute Problem Solver).
  - Una técnica denominada means-ends analysis.

STRIPS (1971)

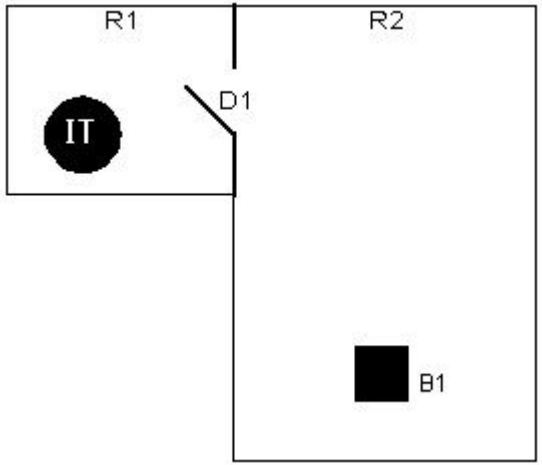
Describimos un problema como:

- Un estado es definido por un conjunto de atributos (*predicados*).
  - Hay un estado Inicial
  - Hay un estado objetivo
- Un conjunto de acciones posibles. Cada acción posee:
  - Un conjunto de precondiciones (atributos necesarios para poderse aplicar)
  - Un conjunto de post-condiciones (cómo quedan los atributos luego de aplicar la acción)
- STRIPS busca una secuencia de acciones que lleva del estado inicial al objetivo
  - Las precondiciones se deben satisfacer en cada paso.
  - Parecido a buscar un camino en un grafo.

STRIPS en un robot: predicados

El conocimiento del robot puede estar representado por los siguientes predicados:

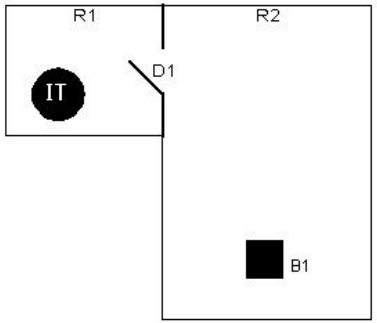
- **INROOM(x,r)**, donde:
  x es un objeto de tipo *movable_object*
  r es de tipo *room*.
- **NEXTTO(x,t)**, donde:
  x es un objeto de tipo *movable_object*
  t es un objeto de tipo *movable_object* o *door*.
- **STATUS(d,s)**, donde:
  d es de tipo *door*
  s es un enumerado de tipo: *OPEN* o *CLOSED*.
- **CONNECTS(d,rx,ry)**, donde:
  d es de tipo *door*
  rx,*ry* son de tipo *room*



STRIPS en un robot: estados

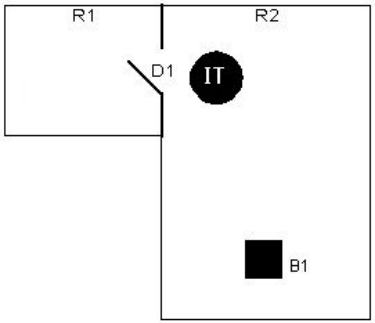
Estado inicial:

INROOM(IT,R1)
INROOM(B1,R2)
CONNECTS(D1,R1,R2)
CONNECTS(D1,R2,R1)
STATUS(D1,OPEN)



Estado final:

INROOM(IT,R2)
INROOM(B1,R2)
CONNECTS(D1,R1,R2)
CONNECTS(D1,R2,R1)
STATUS(D1,OPEN)



STRIPS en un robot: acciones

Tabla parcial de diferencias:

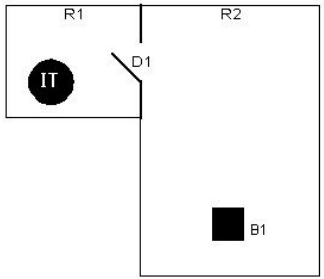
Operador	Precondición	add-list	delete-list
OP1:GOTODOOR(IT,dx)	INROOM(IT,rk) CONNECTS(dx,rk,rm)	NEXTTO(IT,dx)	
OP2:GOTHRUDOOR(IT,dx)	CONNECTS(dx,rk,rm) NEXTTO(IT,dx) STATUS(dx,OPEN) INROOM(IT,rk)	INROOM(IT,rm)	INROOM(IT,rk)

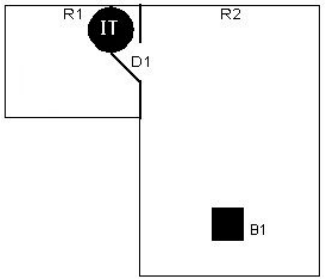
Si queremos aplicar OP2:

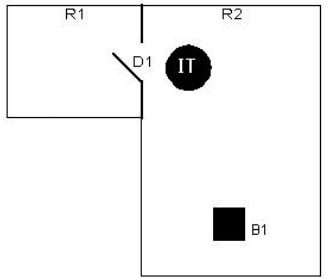
1. Instanciamos parámetros
2. Verificamos precondiciones
3. Agregamos y borramos de listas de predicados

STRIPS en un robot: plan de acción

Estado inicial	Estado luego de aplicar el OP1	Estado luego de aplicar el OP2
INROOM(IT,R1) INROOM(B1,R2) CONNECTS(D1,R1,R2) CONNECTS(D1,R2,R1) STATUS(D1,OPEN)	INROOM(IT,R1) INROOM(B1,R2) CONNECTS(D1,R1,R2) CONNECTS(D1,R2,R1) STATUS(D1,OPEN) NEXTTO(IT,D1)	INROOM(IT,R2) INROOM(B1,R2) CONNECTS(D1,R1,R2) CONNECTS(D1,R2,R1) STATUS(D1,OPEN) NEXTTO(IT,D1)





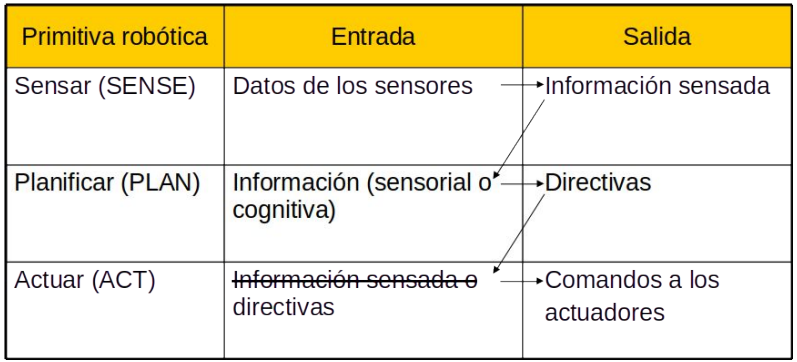


STRIPS en un robot: plan de acción

Para encontrar un plan, se evalúan recursivamente acciones que reducen la *distancia* (predicados incompatibles) entre el estado actual y el final.

Asume que toda la información necesaria para resolver el problema está descrita (Closed World Assumption)

Paradigma Jerárquico:



STRIPS en un robot: discusión

Problemas comunes en sistemas jerárquicos

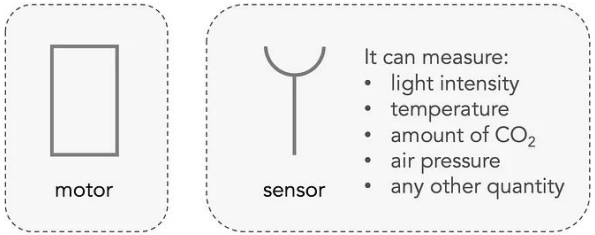
- Frame Problem:
  - El problema de representar la situación del mundo real de forma que sea computacionalmente tratable.
  - Cómo determinar eficientemente que las cosas siguen siendo iguales en un mundo que cambia.
- Cualquier problema involucra todo lo que se conoce.
- Un robot real puede necesitar varios planes de escala y alcance distinto, ¿cómo combinarlos?
- ¿Cómo manejar la incertidumbre semántica, del sensado, y de la actuación?

Caso de estudio: *Braitenberg Vehicles*

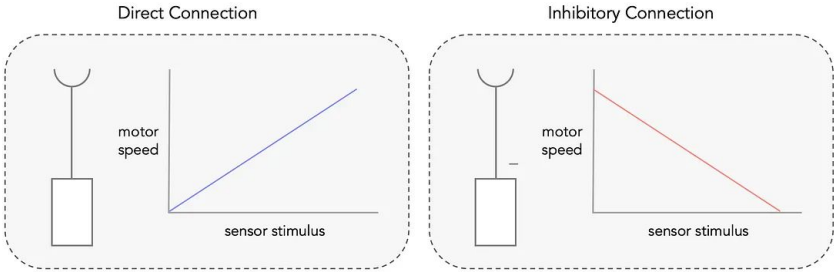
Braitenberg Vehicles (1986)

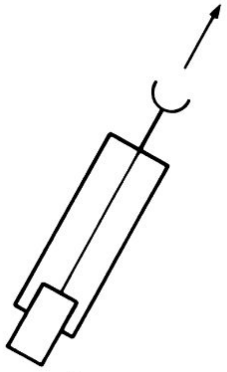
Planteado como un “experimento mental”, es fácil de implementar.

- Disponemos de:



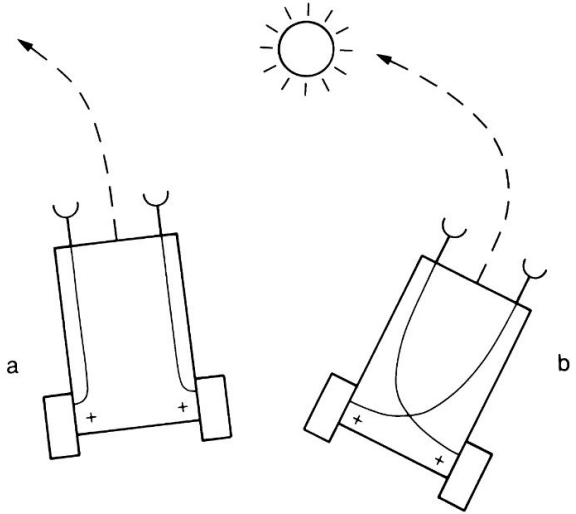
- Los podemos conectar:



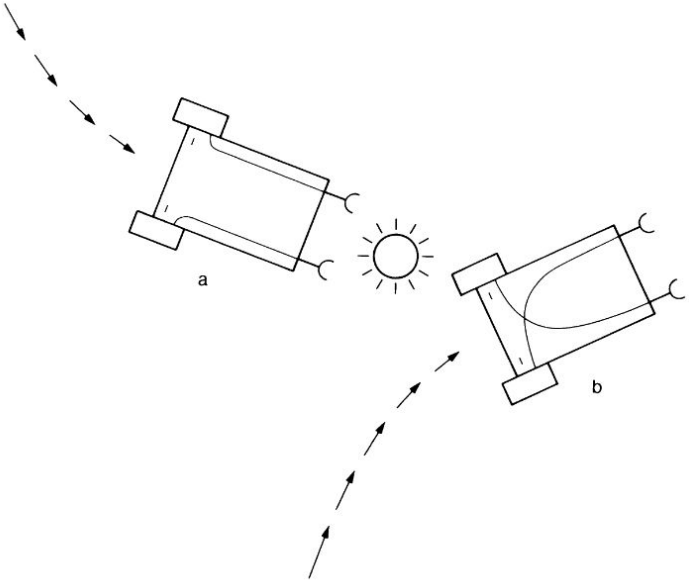


Vehículo 1

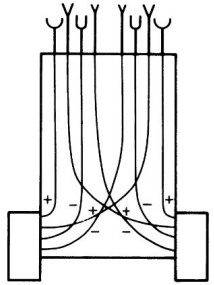
Braitenberg Vehicles



Vehículos 2a (Fear) y 2b (Agression)



Vehículos 3a (Love) y 3b (Explorer)



Vehículos 3c (multisensorial)

Braitenberg Vehicles

- Dispositivo analógico
- Combinación de múltiples sistemas concurrentes

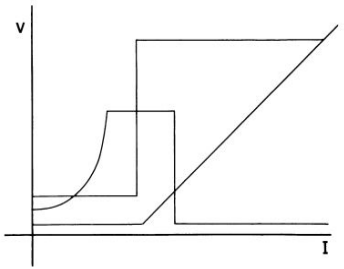


Figure 8
Various bizarre kinds of dependence of the speed of the motor (ordinate) on the intensity of stimulation (abscissa) in Vehicle 4b.

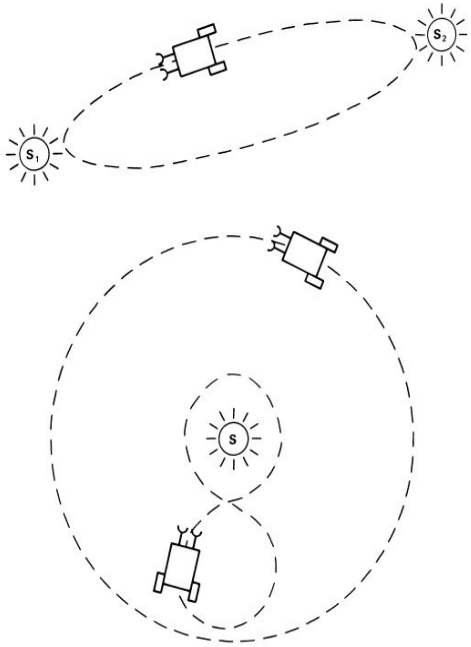


Figure 7
Trajectories of vehicles of brand 4a around or between sources.

Braitenberg Vehicles: discusión

Características comunes en Paradigma reactivo:

- Inspiración biológica
- No hay “controladores”: Comportamiento Emergente
- Buenos tiempos de respuesta
- No tiene memoria
- No tiene modelo del mundo
- Implica superposición de acciones
- Modularizable (?)

Primitiva robótica	Entrada	Salida
Sensar (SENSE)	Datos de los sensores	Información sensada
Actuar (ACT)	Información sensada o directivas	Comandos a los actuadores

Caso de estudio: campos de potencial

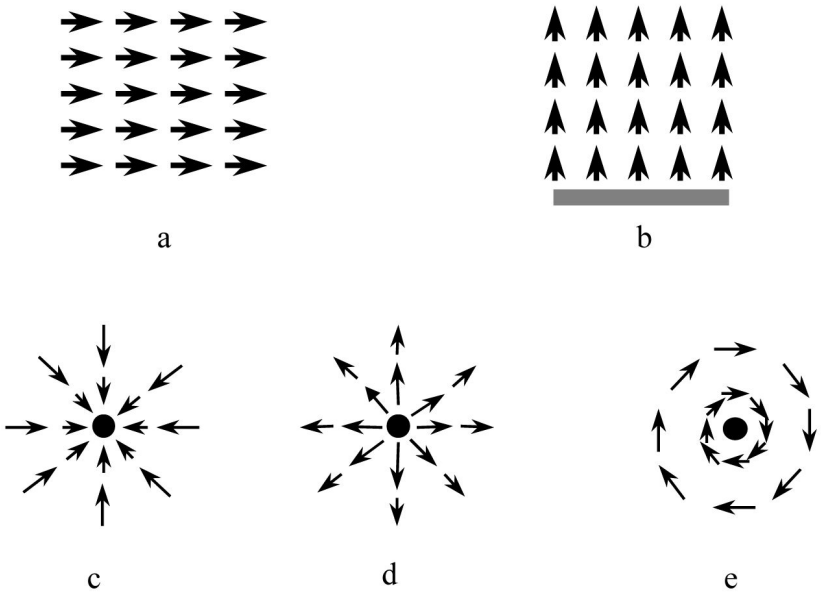
Campos de potencial (Brooks, 1986)

Mecanismo para describir influencia del sensado en la actuación.

- Una acción es un campo de potencial, o un campo de fuerzas.
- El efecto sobre el robot es la fuerza muestreada en su posición.
- El campo se construye a partir del sensado. (nota mental)
- El campo se construye como la suma de campos fundamentales (primitivas)

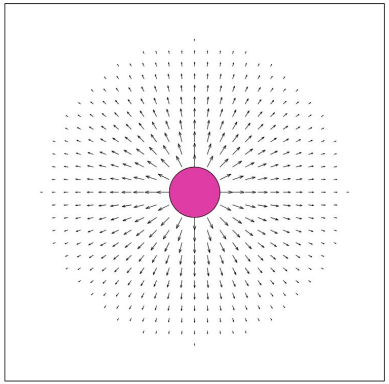
Campos de potencial: primitivas

- Uniforme
  - Moverse en una dirección particular, seguir un corredor.
- Perpendicular
  - Evitar paredes
- Atracción
  - Moverse hacia el objetivo
- Repulsión
  - Evitar obstáculos
- Tangencial
  - Moverse a través de una puerta
- Aleatorio
- Además un “perfil de magnitud”

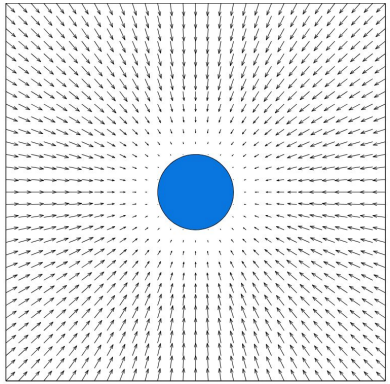


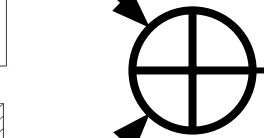
Campos de potencial: primitivas

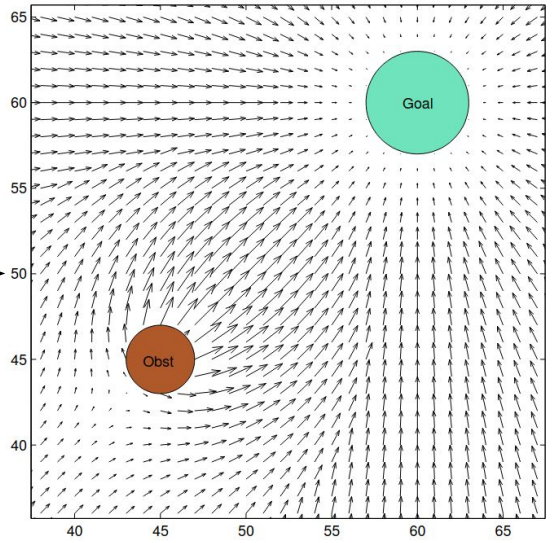
Obstáculo
(repulsión)



Objetivo
(atractor)

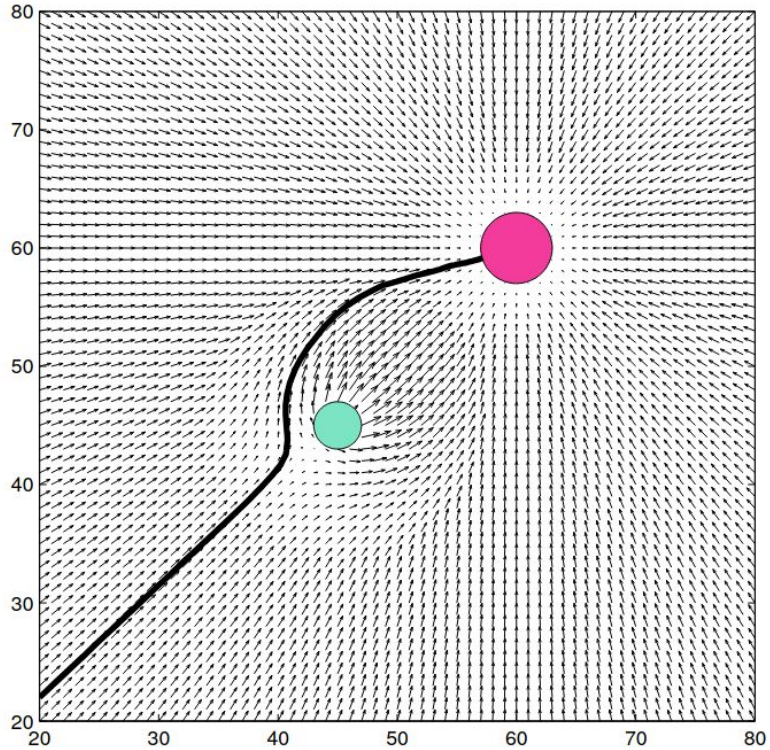






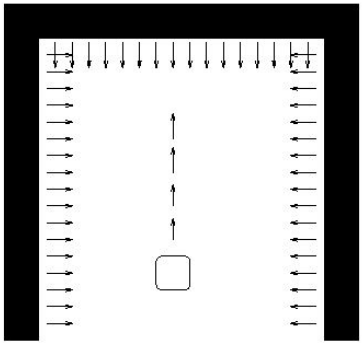
> Goodrich M. A., *Potential Fields Tutorial*

Campos de potencial: ir a un objetivo



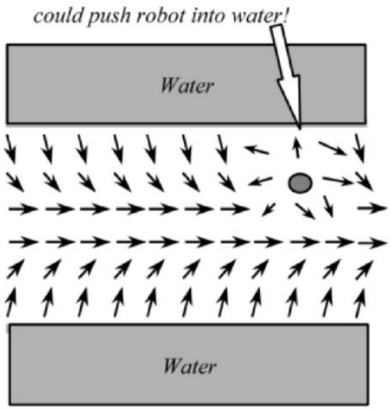
Campos de potencial: discusión

- Se puede visualizar fácilmente.
- ¿Qué pasa con lo de “Parcialmente Observable”?
- Fácil de implementar.
- Como en los *vehículos*, las primitivas (comportamientos) sólo pueden sumarse.



Problemas comunes a sistemas reactivos:

- La combinación de varias primitivas puede dar 0.
- Movimientos espasmódicos (oscilaciones).
- Mínimos locales.
- Solo aplicable a navegación o control de movimientos.



Caso de estudio: máquinas de estado

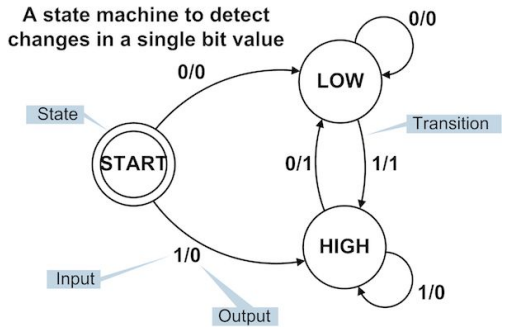
Máquinas de estado

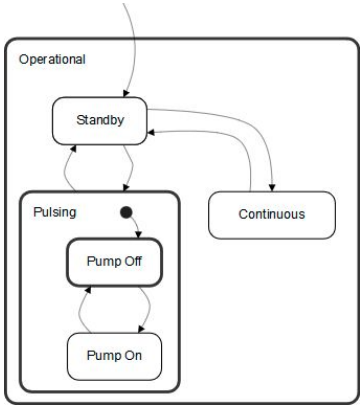
Máquina de Estado Finito (FSM), una abstracción clásica para describir el comportamiento de un sistema:

- Sistema definido por un conjunto de estados y transiciones posibles entre estados. Cada transición es activada por un evento, y dispara una acción.
- Hay un estado inicial, y puede haber uno final (“estado absorbente”)
- Nuevo concepto de “Sin Memoria”: proceso Markoviano.

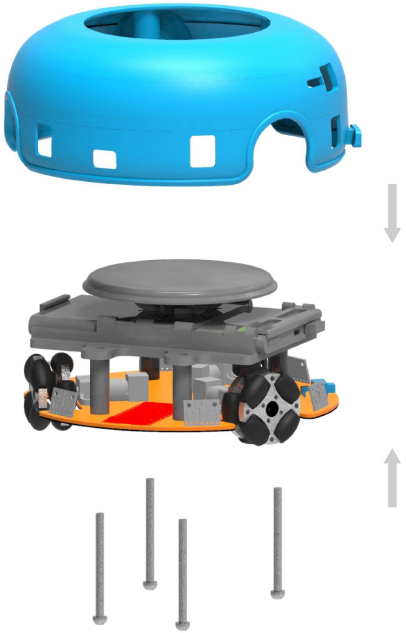
Nos interesa un tipo particular: Máquinas de Estado Jerárquicas (HSM)

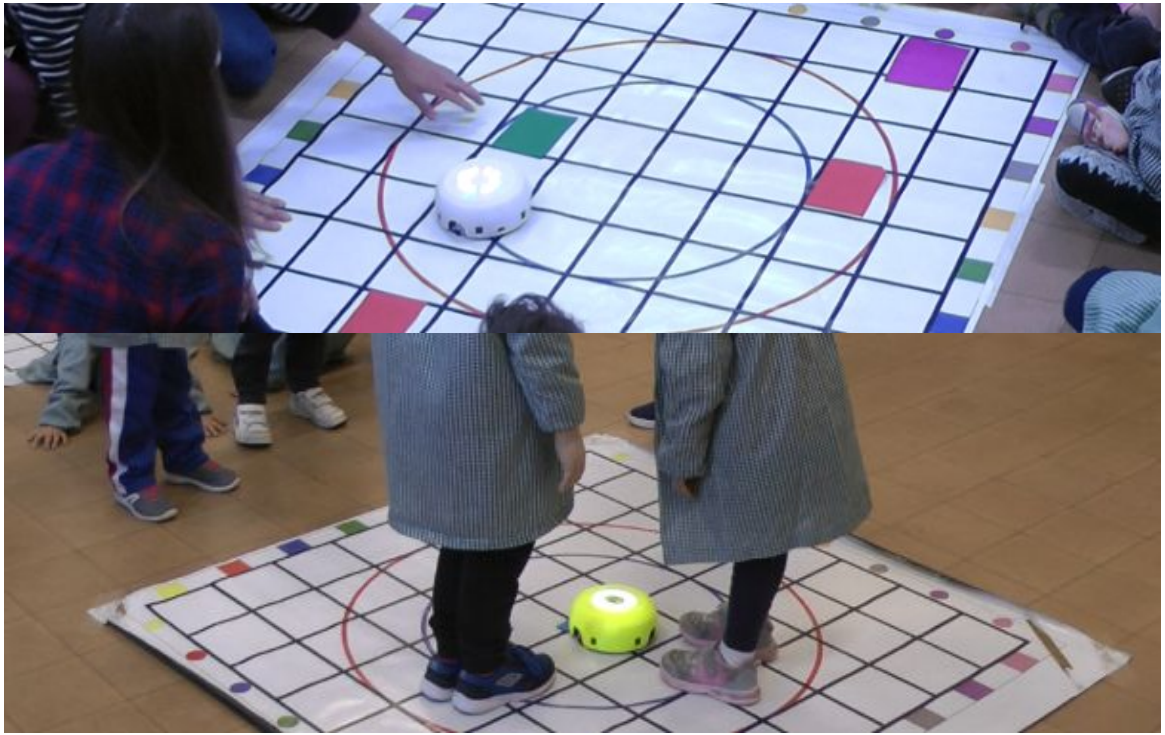
- Un estado puede tener asociada una HSM que está activa mientras estamos en ese estado (definición recursiva)





Robotito + HSM



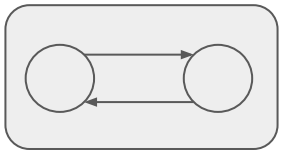


Robotito + HSM

Robotito es un robot microcontrolado

- Sensores de piso, de color y de distancia
- Tiene un intérprete de Lua, un lenguaje embebido. Permite *scriptear* el robot.
- *ahsm*, una biblioteca embebida en Lua.

Ejemplo de comportamiento del robot: se prende al apoyar en el piso, se apaga al levantar.



```
--declare states
roboton = ahsm.state {
  entry = omni.enable -- a function to be called on entry
}
robotoff = ahsm.state {
  entry = omni.disable
}

--declare transitions
to_on = ahsm.transition {
  src=robotoff, tgt=roboton, events={'onfloor'}
}
to_off = ahsm.transition {
  src=roboton, tgt=robotoff, events={'notfloor'}
}

-- create handler that emits floor events
proximity.cb.append( function( on_floor )
       robot.hsm.event(on_floor and 'onfloor' or 'notfloor')
  end )

-- root state, that contains main machine
local hsm = ahsm.state {
  states = {roboton, robotoff},
  transitions = {to_on, to_off},
  initial = robotoff -- initial state
}

return hsm
```

> https://github.com/xopxe/ahsm
> https://gitlab.fing.edu.uy/robotito/firmware

Robotito + HSM

Puedo combinar máquinas de estado, embebiendo unas en otras para detallar el comportamiento correspondiente a cada estado. En el ejemplo anterior, qué hacer cuando estamos en “modo on”?

We developed two behaviors that allow to interact with the robot. One uses the color sensor to indicate a direction to move. The other uses the distance sensor to allow the robot to react to objects. More interaction modes are being developed.

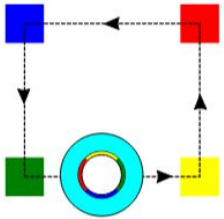


Fig. 4. Controlling Robotito with colors.

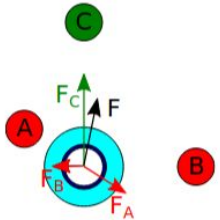
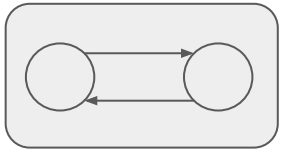


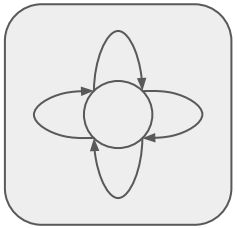
Fig. 5. Controlling Robotito with forces.

> Designing child-robot interaction with Robotito - RO-MAN2019

Robotito + HSM

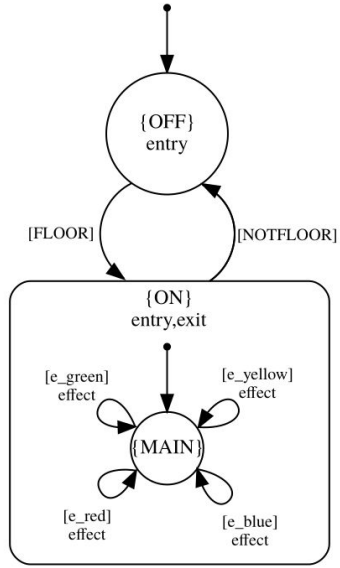


onoff.lua



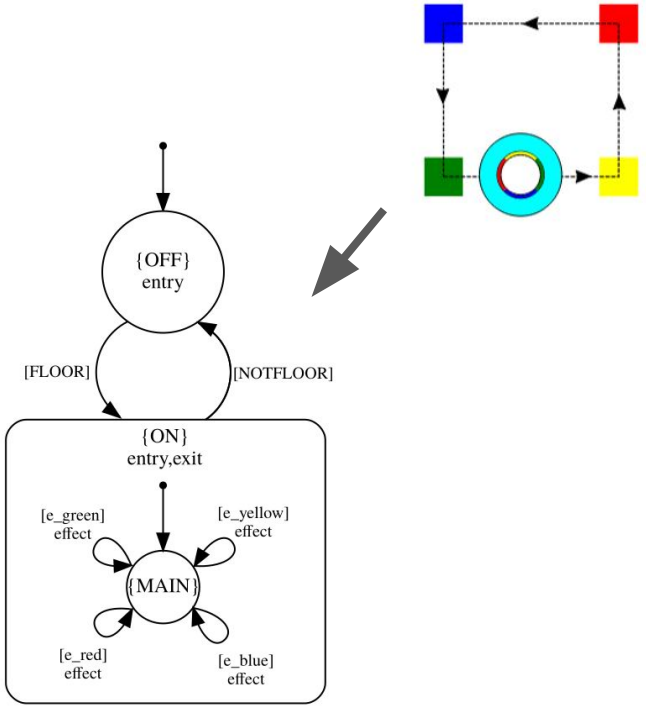
color.lua

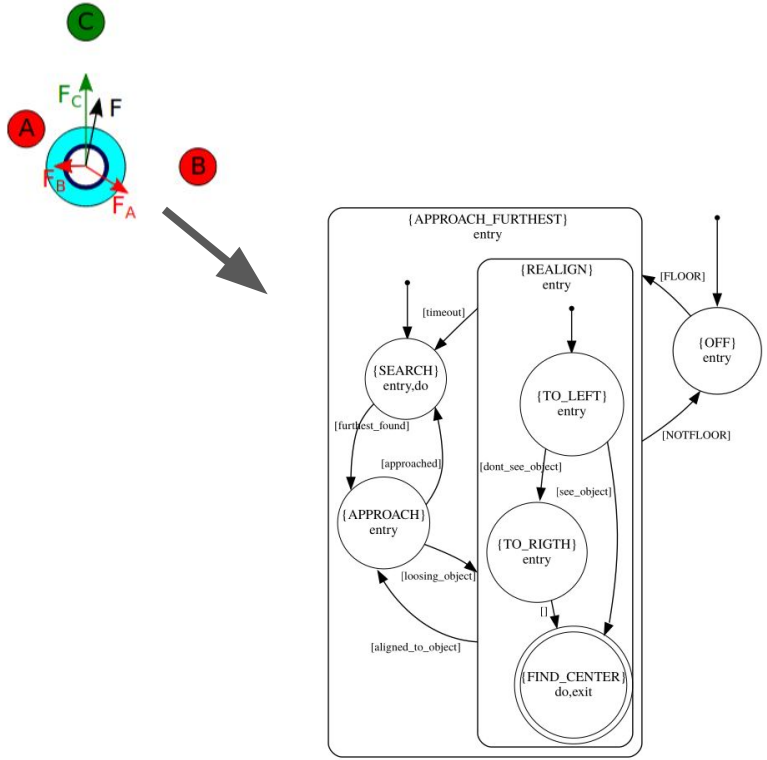
```
-- load state machine
mionoff = require 'states.onoff'
move = require 'states.colormove'

-- add robot enabling/disabling
mionoff.states.ON.entry = function()
  robot.omni.enable(true)
end
mionoff.states.ON.exit = function()
  robot.omni.enable(false)
end

-- embed state machine
mionoff.states.ON.states = {move}
mionoff.states.ON.initial = move

return mionoff
```



mionoff.lua

Robotito + HSM





Caso de estudio: *Subsumption*

Subsumption (Brooks, 1985)

Un comportamiento es una red de módulos de sensado y actuación que realizan una tarea.

Los módulos se agrupan en capas de competencias.

Los módulos de una capa pueden *sobreescribir* o *subsumar* la salida de un comportamiento de una capa inferior.

No utiliza mantiene ningún modelo del mundo.

Acknowledgments. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research is provided in part by an IBM Faculty Development Award, in part by a grant from the Systems Development Foundation, in part by an equipment grant from Motorola, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

Subsumption (Brooks, 1985)

Jerárquico vs Reactivo

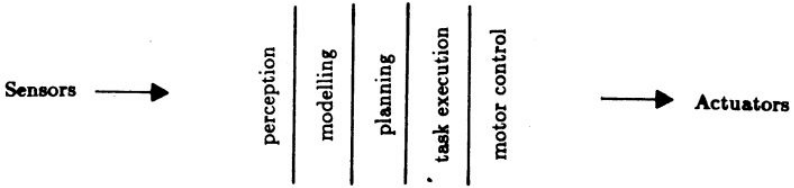


Figure 1. A traditional decomposition of a mobile robot control system into functional modules.

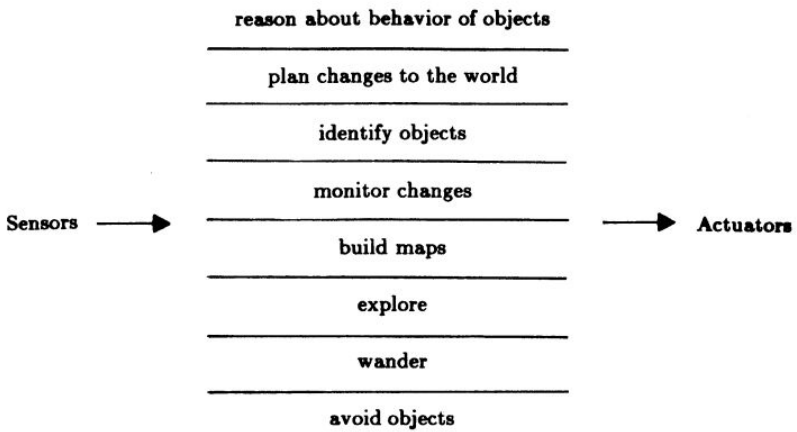
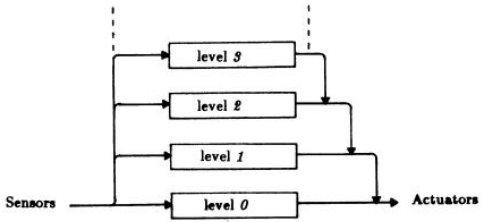


Figure 2. A decomposition of a mobile robot control system based on task achieving behaviors.

Subsumption: capas

Cada capa es una capacidad (competencia) completa.



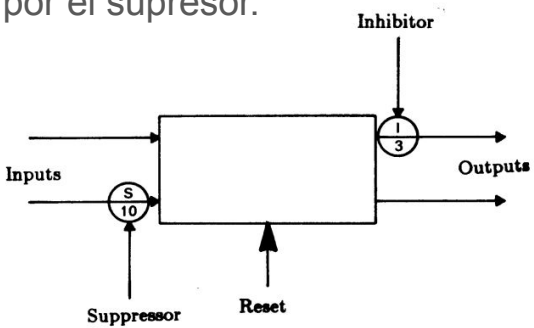
- Capas de competencia para un robot móvil:
  0. Evitar colisiones.
  1. Deambular sin un objetivo (*wander*).
  2. Explorar: detectar lugares remotos e ir hacia ellos.
  3. Construir un mapa del entorno y planificar rutas.
  4. Detectar cambios en el entorno.
  5. Razonar sobre el mundo en términos de objetos identificable.
  6. Formular y ejecutar planes que impliquen manipular el mundo.
  7. Razonar sobre el comportamiento de los objetos y ajustar los planes.

  (¿por qué estamos en la sección “reactivos”?)
- Se componen de módulos que se comunican.
- Se puede desarrollar independientemente de las superiores.
- Todas se alimentan del sensado.
- Una capa puede manipular a una capa inferior (a nivel de módulos).

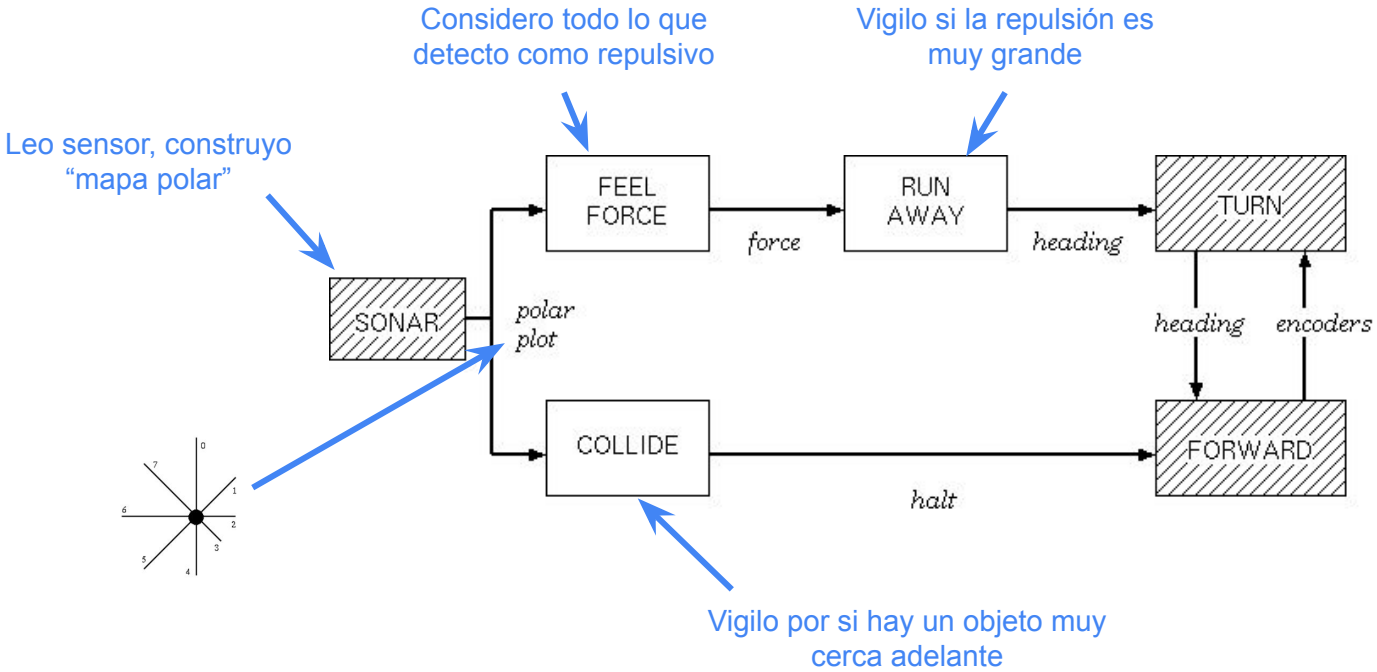
Subsumption: módulos

Los módulos son la unidad de programa

- En la visión original es una AFSN escrita en LISP
- Los módulo intercambian señales para pasarse información
  - Las “interconexiones” entre módulos son fijas
- Además una señal puede:
  - **Inhibir** una salida: la salida queda apagada por un tiempo luego de llegar una señal.
  - **Suprimir** una entrada: los datos de entrada son reemplazados por el supresor.



Nivel 0: *Runaway*

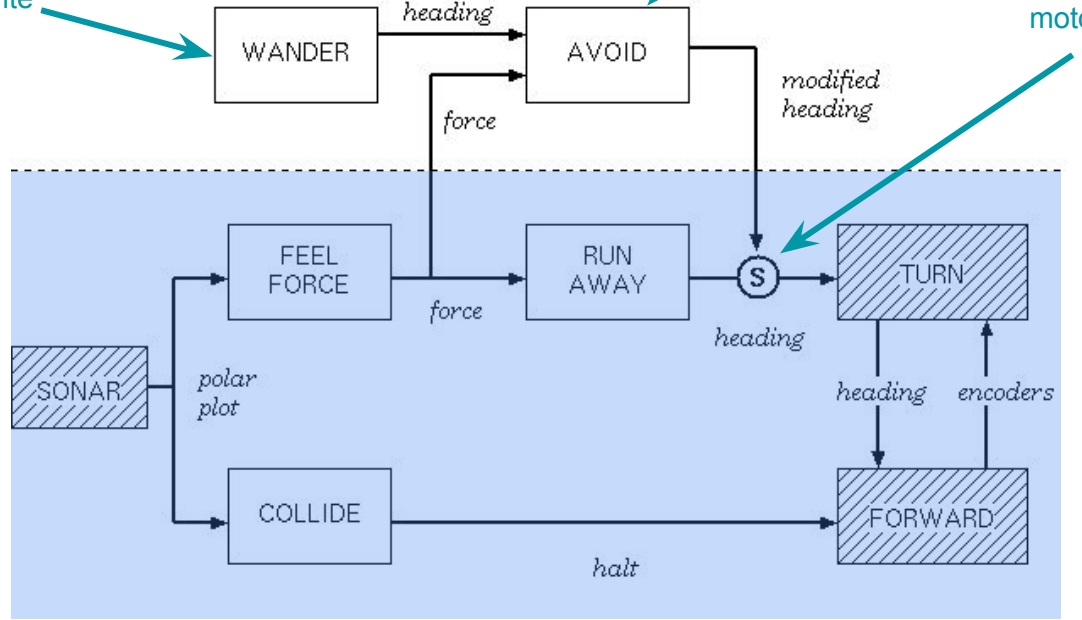


Nivel 1: *Wander*

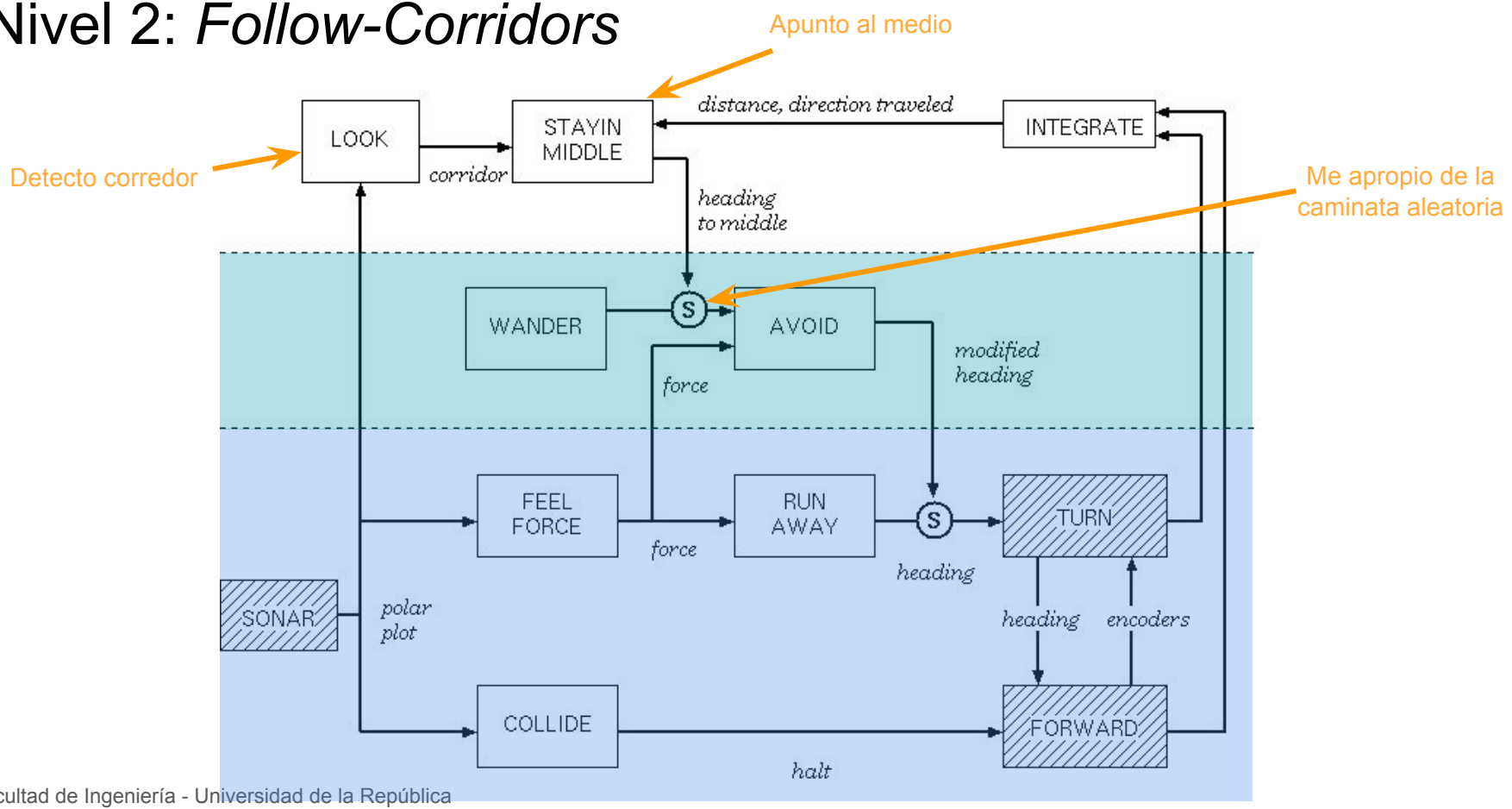
Combino fuerzas ponderando la dirección deseada

Genero una dirección aleatoria periódicamente

Me apropio de los motores (suprimo)



Nivel 2: *Follow-Corridors*



Subsumption: discusión

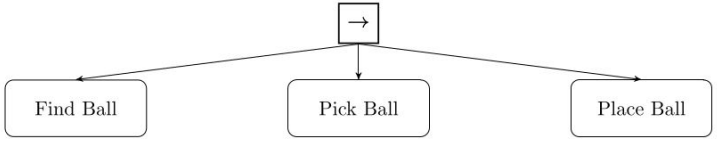
- Muy difícil llevarlo más allá de la capa 2 (explorar).
  - La complejidad explota al subir de nivel
  - Modularización “plana”, interifirendo entre capas. Cambios en una capa pueden romper otras.
  - La jerarquía de las capas puede ser arbitraria (puede interesar subsumar “hacia arriba”)

Caso de estudio: *behavior trees*

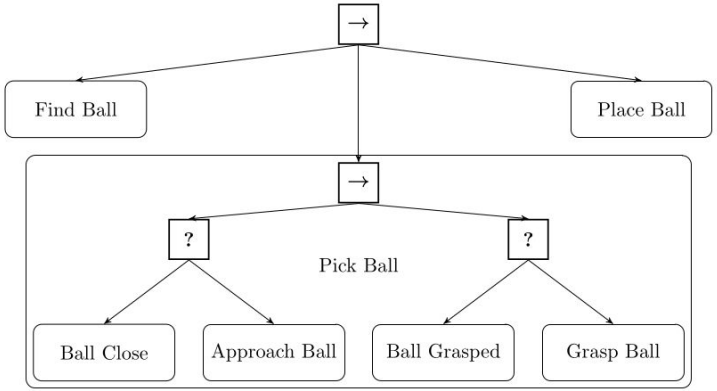
Behavior trees

- Originalmente desarrollados en la industria de los juegos, para especificar el comportamiento de NPCs
- Sirven una función parecida a las Máquinas de Estado, pero de forma más modular.
- El comportamiento se especifica como un árbol con dos tipos de nodos:
  - Nodos internos: “*control flow nodes*”, controlan la lógica de ejecución
  - Nodos hoja: “*execution nodes*”, acciones.
- El árbol es recorrido periódicamente en DFS, respetando los “*control flow nodes*”, y ejecutando los “*execution nodes*” que se visitan.
  - Al ser visitados los nodos devuelven:
    - *Running* cuando están activos
    - *Success* o *Failure* cuando terminan

Behavior trees



(a) A high level BT carrying out a task consisting of first finding, then picking and finally placing a ball.



(b) The Action Pick Ball from the BT in Fig. 1.1a is expanded into a sub-BT. The Ball is approached until it is considered close, and then the Action grasp is executed until the ball is securely grasped.

> Michele Colledanchise,Petter Ögren - "*Behavior Trees in Robotics and AI*"

Behavior trees: control flow nodes (1/4)

Sequence

Activa los hijos en orden, hasta que:

- Uno devuelve *Failure* o *Running*, entonces devuelve este estatus al invocador
- Llega hasta el final, entonces devuelve *Success*
- Se parece a un *Short-circuit AND*

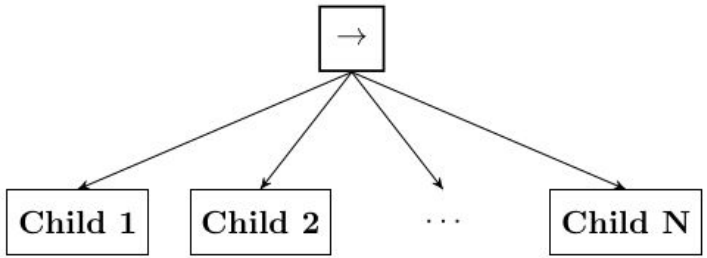


Fig. 1.2: Graphical representation of a Sequence node with *N* children.

Behavior trees: control flow nodes (2/4)

Fallback

Activa los hijos en orden, hasta que:

- Uno devuelve *Success* o *Running*, entonces devuelve este estatus al invocador
- Llega hasta el final, entonces devuelve *Failure*
- Se parece a un *Short-circuit OR*

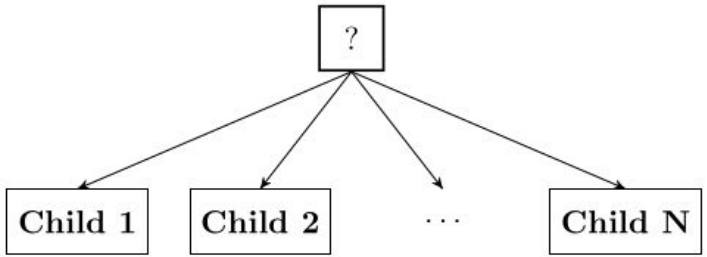


Fig. 1.3: Graphical representation of a Fallback node with *N* children.

Behavior trees: control flow nodes (3/4)

Parallel

Activa los hijos en orden, y luego:

- Devuelve *Success* si al menos M hijos devolvieron *Success*
- Devuelve *Failure* si al menos N-M+1 hijos devolvieron *Failure*
- Si no, devuelve *Running*
- (M es un parámetro del nodo)

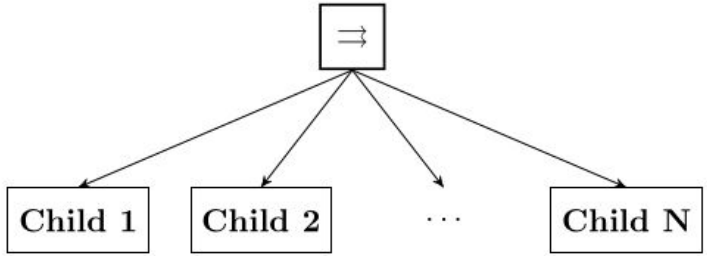


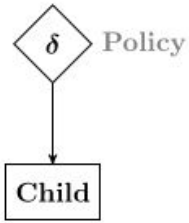
Fig. 1.4: Graphical representation of a Parallel node with *N* children.

Behavior trees: control flow nodes (4/4)

Decorator

Aplica una operación sobre los retornos de sus hijos.
Por ejemplos:

- Invert: invierte *Failure* / *Success*
- max-N-tries: si el hijo devuelve *Failure* N veces, deja de visitarlo y devuelve *Failure*.
- max-T-sec: si el hijo devuelve *Running* durante más de T segundos, deja de visitarlo y devuelve *Failure*



(c) Decorator node. The label describes the user defined policy.

Behavior trees: execution nodes (1/2)

Condition

Contiene una proposición

- Si la proposición es verdadera, devuelve *Success*
- Si la proposición es falsa, devuelve *Failure*
- No devuelve *Running*

(b) Condition node. The label describes the condition verified.

Behavior trees: execution nodes (2/2)

Action

Ejecuta una acción. El árbol hace polling:

- Si la acción tuvo éxito, devuelve *Success*
- Si la acción fracasó, devuelve *Failure*
- Si la acción continúa, devuelve *Running*

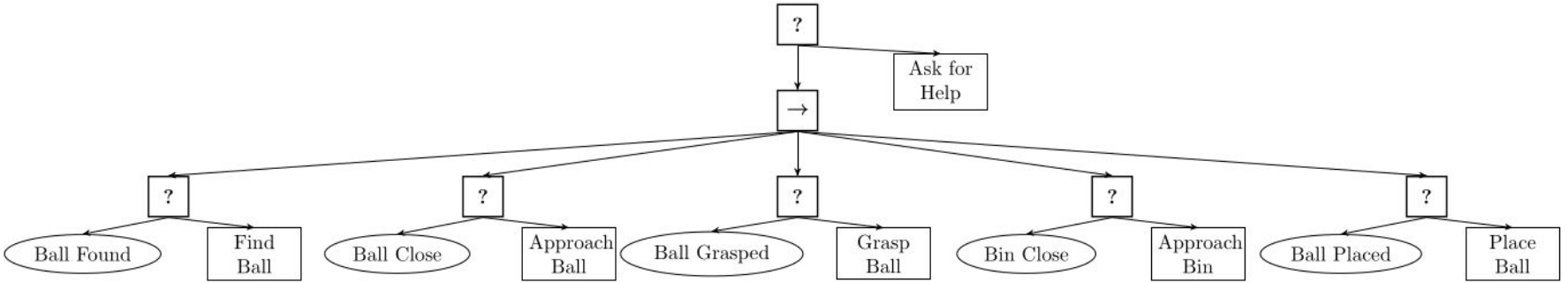
(a) Action node. The label describes the action performed.

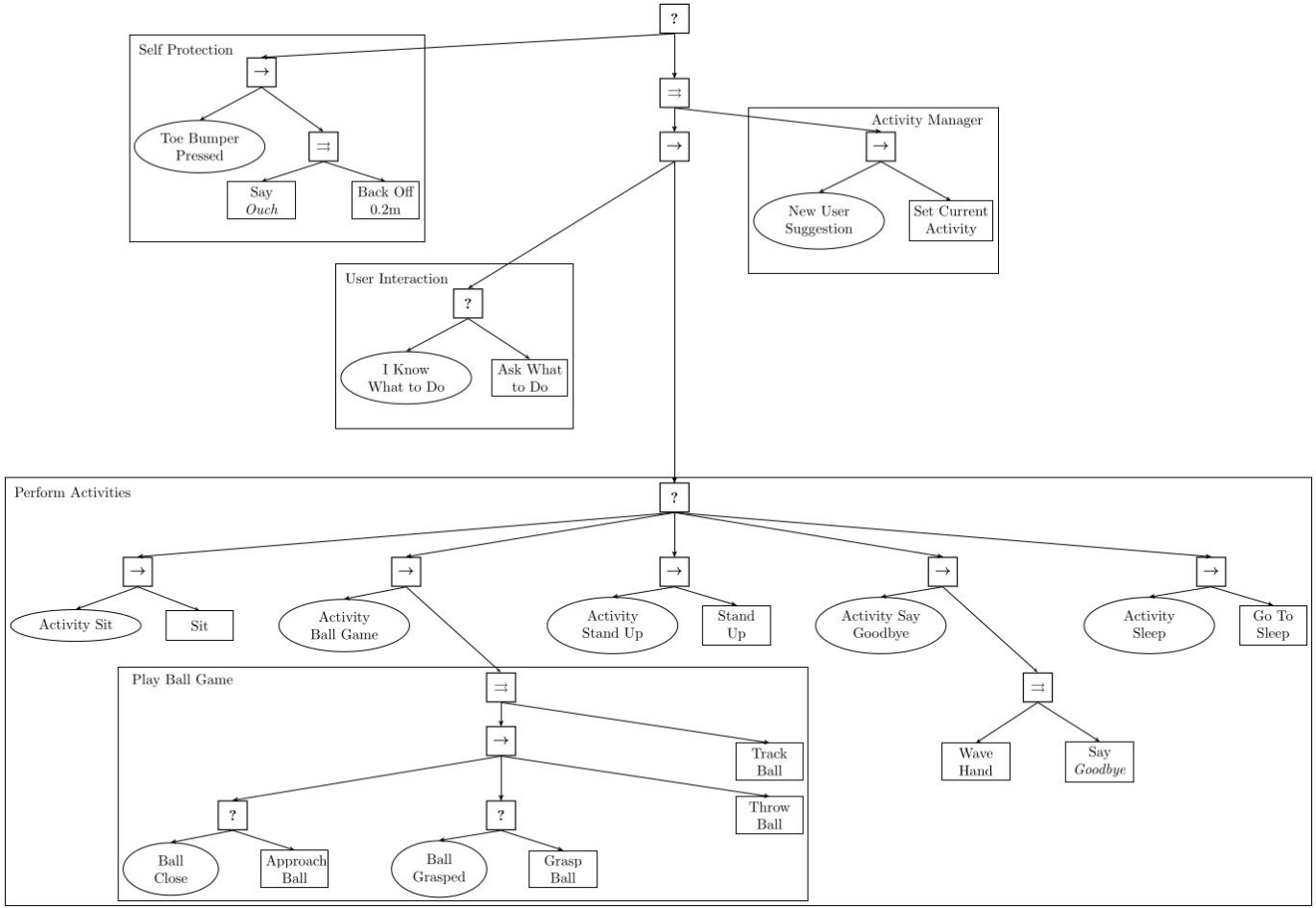
Behavior trees: resumen de nodos

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	$\rightarrow$	If all children succeed	If one child fails	If one child returns Running
Parallel	$\rightrightarrows$	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	$\Diamond$	Custom	Custom	Custom

Behavior trees: ejemplo

Encontrar una pelota, agarrarla y llevarla a un canasto. Responder a fallos.




?
Self Protection
→
Toe Bumper Pressed
⇉
Say *Ouch*
Back Off 0.2m
⇉
→
Activity Manager
→
New User Suggestion
Set Current Activity
User Interaction
?
I Know What to Do
Ask What to Do
Perform Activities
?
→
Activity Sit
Sit
→
Activity Ball Game
Play Ball Game
⇉
→
Track Ball
Throw Ball
?
Ball Close
Approach Ball
?
Ball Grasped
Grasp Ball
→
Activity Stand Up
Stand Up
→
Activity Say Goodbye
⇉
Wave Hand
Say *Goodbye*
→
Activity Sleep
Go To Sleep

Behavior trees: discusión

- HFSM y BT proveen modularización y reactividad.
- Las transiciones en HFSM son parecidas a un GOTO: el salto es sin retorno.
- Las transiciones en BT son parecidas a una llamada a una función: se entra, y luego se vuelve con un retorno.
- Sugiere ser más estructurado, reusable, modular. Probablemente escale mejor a comportamientos complejos.
- Los BTs en sí son relativamente complejos de implementar. Por ejemplo, ¿cómo integrar eventos?

Discusión: jerárquico vs reactivo

Síntesis

Jerárquico

- Tiene un modelo: puede razonar sobre el pasado y proyectarse en el futuro.
- Control de la semántica del modelo.
- Ciclo de vida lento (tiempo de vida de los planes).
- Poder: podemos encontrar soluciones óptimas
- Problema: el modelo interno se desincroniza con la realidad

Reactivo

- No tiene memoria: sólo considera el presente.
- Comportamiento emergente.
- Ciclo de vida rápido (reacción).
- Poder: no almacenamos nada que pueda ser mentira.
- Problema: dependemos unicamente del sensado, que es local (y puede mentir).

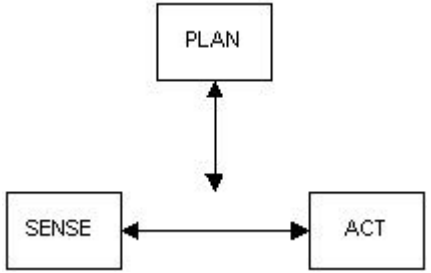
Paradigma híbrido

Bajo este paradigma el robot:

1. Planifica cómo descomponer la tarea en subtareas.
2. Determina cuáles son los comportamientos adecuados para realizar las subtareas.
3. Ejecuta los comportamientos adecuados para cada subtarea.

Preguntas:

- ¿Cómo distingue la arquitectura entre reacción y deliberación?
- ¿Cómo organiza las responsabilidades dentro de la parte deliberativa?
- ¿Cómo emerge el comportamiento global?



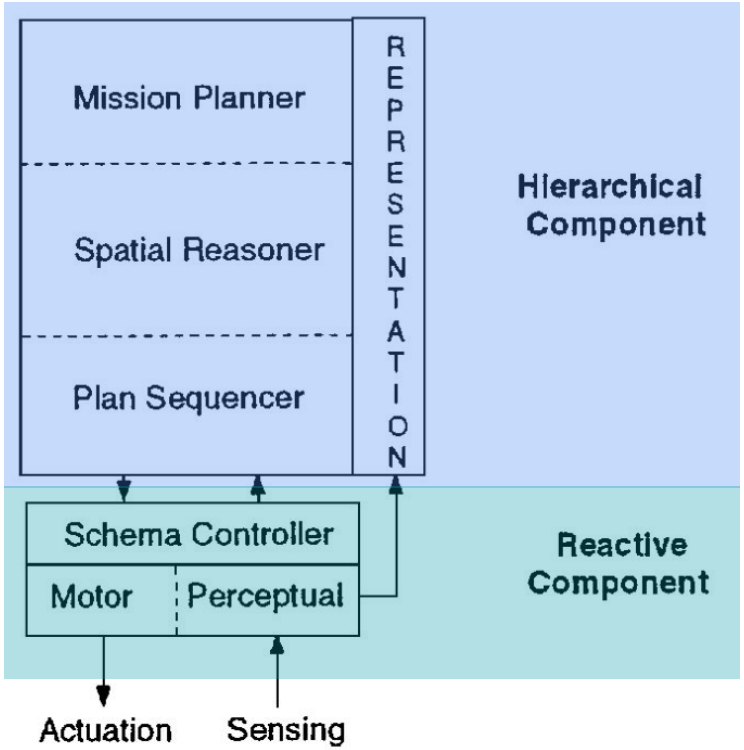
Primitiva robótica	Entrada	Salida
Planificar (PLAN)	Información (sensorial o cognitiva)	Directivas
Sensar - Actuar (S-A)	Datos de los sensores	Comandos a los actuadores

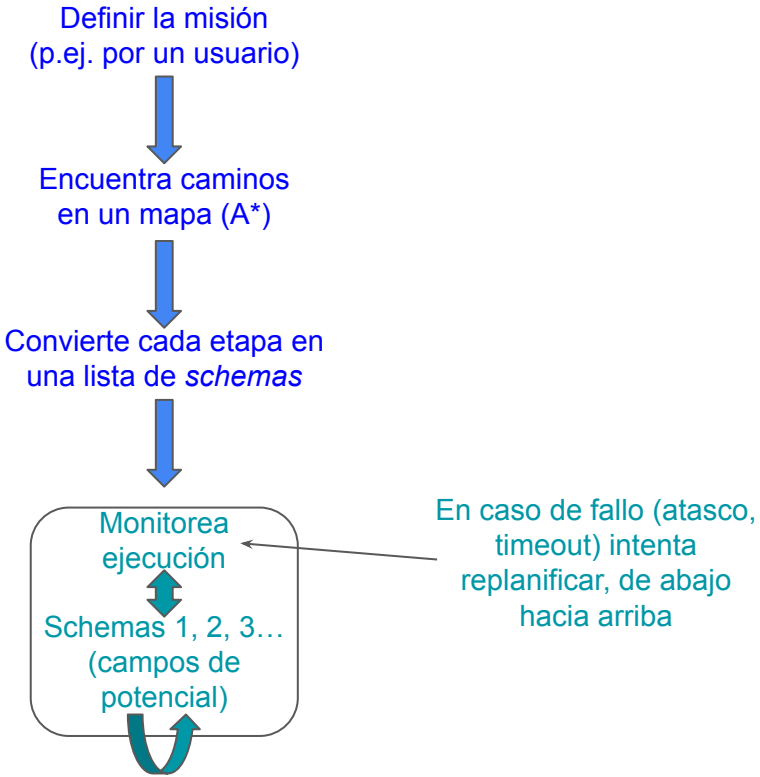
Caso de estudio: AuRA

AuRA: Autonomous Robot Architecture (Arkin, 1986)

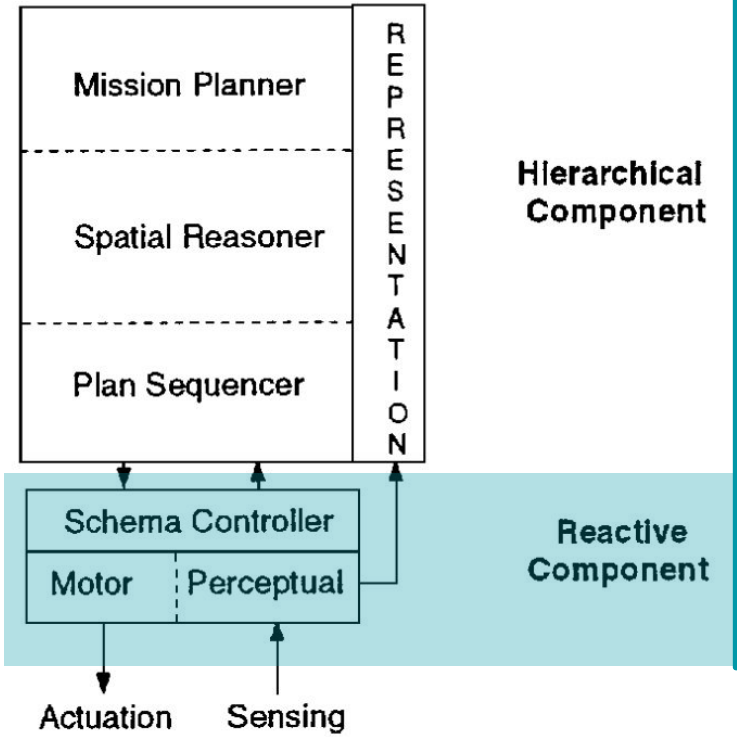
- De los primeros sistemas del paradigma híbridos.
- Componentes reactivos y jerárquicos.
- Inspirado en conceptos de la biología.
- El componente reactivo está basado en la Teoría de los Esquemas.
  - Patrones de percepción, dedicados a procesar información sensorial (p. ej. fusión)
  - Patrones motrices, dedicados a generar actuación.
  - Se combinan usando Campos de Potencial.
- Introduce el concepto de control homeostático.

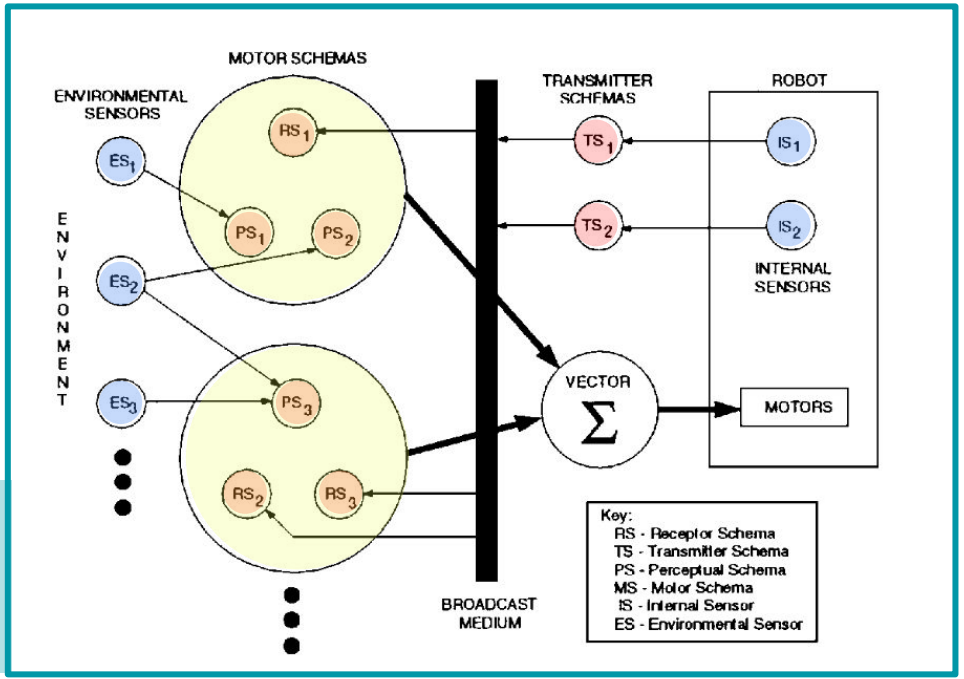
AuRA: Autonomous Robot Architecture (1986)





AuRA: Autonomous Robot Architecture (1986)





Caso de estudio: *Three-layer architecture*

Three-layer architecture (90s)

Derivado de la experiencia al desarrollar robots móviles. Agrupamos los algoritmos según su cómo usan modelos:

- Basados en sensores, sin modelo: **Controller**
- Usan memoria sobre el pasado: **Sequencer**
- Hacen predicciones sobre el futuro: **Deliberator**

> Gat, E., On Three-Layer Architectures

Three-layer architecture: *1. Controller*

Conjunto de bucles de control (*behaviors*, *skills*) que se pueden activar a demanda. Los comportamientos primitivos deben:

- Ser de tiempo-real, con costo (tiempo, recursos) de cotas constantes.
- Debes detectar (y avisar) su propio fallo al controlar.
- Si no le queda otra que mantener estado (p.ej. para filtrar), este debe caducar en tiempo de cota constante.
- Los controladores deben ser continuos.

Three-layer architecture: *2. Sequencer*

Decide qué comportamientos primitivos están activos en cada momento

- Debe responder a cambios (objetivos alcanzados, fallos reportados de controladores…)
- “Sequenciamento condicional”: incluye metainformación (“conocimiento procedural”) sobre relación entre tareas, concurrencia, y fallos.
- Debe ser relativamente rápido en relación a los posibles cambios del entorno (no mucho más lento que el *controller*).

Three-layer architecture: *3. Deliberator*

Todo lo más caro computacionalmente va acá.

- Debe aceptar que pueden pasar varios pasos del *Sequencer* entre que empieza a calcular y termina.
- Puede producir planes para el *Sequencer*.
- Puede responder consultas del *Sequencer*.

Caso de estudio: MERLIN2

MERLIN2

Arquitectura híbrida para ROS2, inspirada por la organización del cerebro.

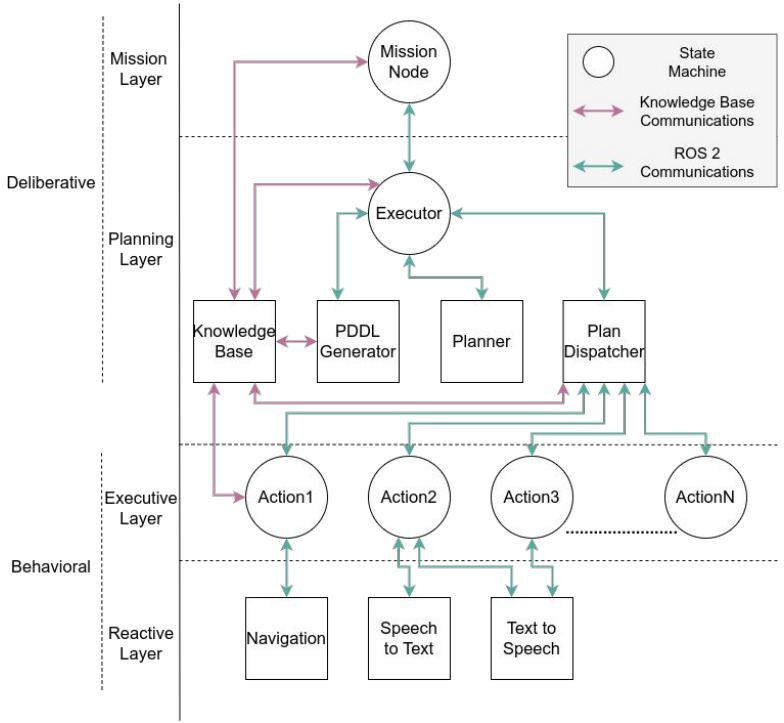
- Desarrollada para robots de servicio.
- Basado en cuatro herramientas:
  - Un motor de FSM (YASMIN)
  - Una base de conocimiento para representación simbólica (KANT)
  - Sistema genérico de planificación
  - Sistema de monitoreo de misión

> https://github.com/MERLIN2-ARCH/merlin2

MERLIN2

Arquitectura de 4 capas:

- Capa de misión: objetivos de alto nivel
- Capa de planificación (subsistema deliberativo)
  - Posee memoria de largo plazo
- Capa ejecutiva: acciones que el robot saber hacer
  - representados como máquinas de estado (memoria de corto plazo con *blackboards*)
- Capa reactiva: colección de *skills*



> https://github.com/MERLIN2-ARCH/merlin2

¿Preguntas?