Control, planificación y optimización

Robótica

Alberto Díaz Álvarez y Raúl Lara Cabrera

Departamento de Sistemas Informáticos - Universidad Politécnica de Madrid

11 de junio de 2023

La teoría del control

Se ocupa del **control de sistemas dinámicos** en procesos de todo tipo.

- Se considera campo interdisciplinario de la ingeniería y de las matemáticas.

¿Cómo **llevar sistemas a estados deseados** en función de sus entradas ...

- ... **minimizando** el **tiempo** de ajuste, rebasamiento y error estacionario?
- ... **garantizando** un nivel de **estabilidad** de control?
- ... **persiguiendo** el grado de **optimalidad**?

Por cierto, las entradas también reciben nombre de **referencia**.

Dentro de la teoría de control existen también otros dos aspectos de estudio:

- **Controlabilidad**: Alterar un sistema usando solo manipulaciones admisibles.
- **Observabilidad**: Medida de lo bien que se infieren los estados internos de un sistema a partir del conocimiento de sus salidas externas.

Existen dos grandes divisiones en la teoría de control, a saber:

- **Clásica**: Diseño de sistemas de una única entrada y una única salida[1].
- **Moderna**: Diseño de sistemas con múltiples entradas y salidas.

[1] Excepto cuando se analiza el impacto de perturbaciones, donde sí se utiliza una segunda entrada.

Función de transferencia

Función que **modela la salida** de un sistema **para cada entrada posible**[2].

El caso más sencillo ofrece una entrada para una salida:

- La gráfica generada se denomina **curva de transferencia**.
- Muy común en áreas como tratamiento de señal o teoría de la comunicación.

Se suele utilizar sólo en sistemas lineales invariantes en el tiempo (LTI):

- La mayoría de sistemas tienen características de entrada/salida no lineales.
- Suelen comportarse linealmente dentro de sus parámetros "normales".

Modelización **teórica**, por lo que no tiene por qué replicar exactamente todos los detalles del sistema modelado.

Ingeniería automática

Puede definirse como la **aplicación práctica de la Teoría del control**.

Sus objetivos fundamentales son:

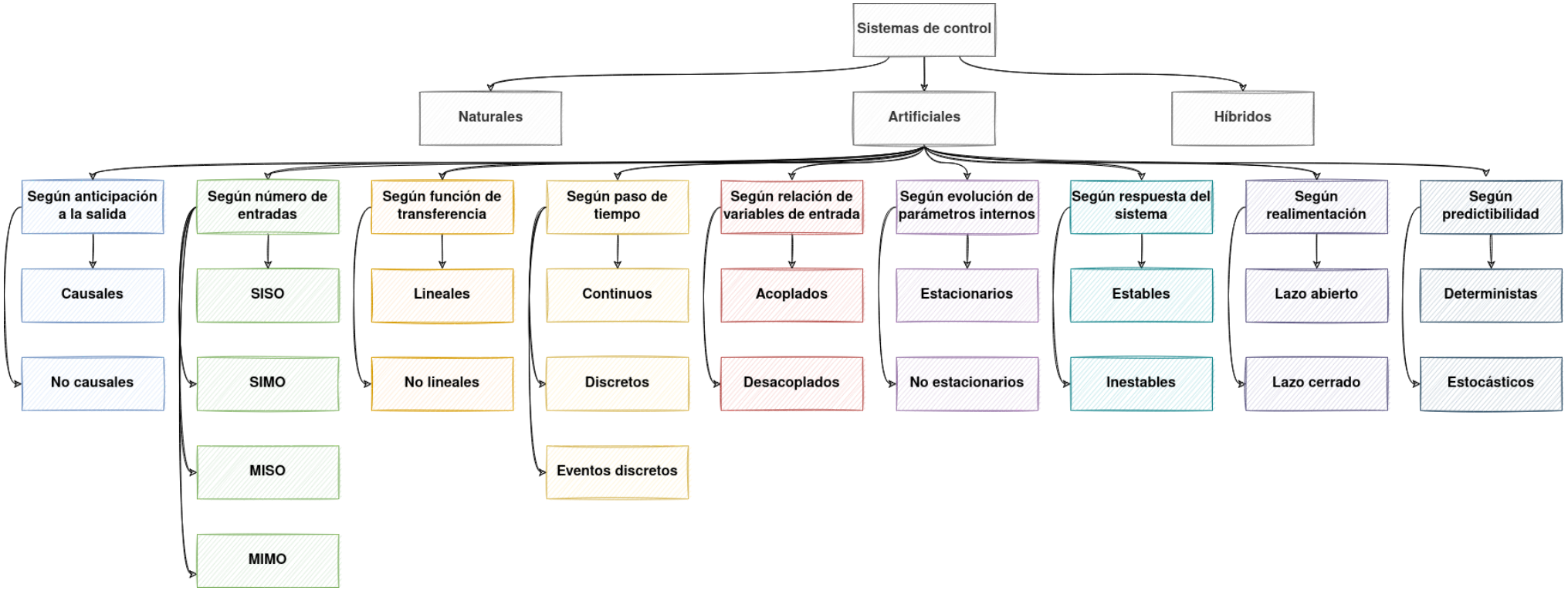
1. **Modelado** de sistemas dinámicos en términos de entradas y salidas.
2. **Diseño** de controladores para regular el comportamiento de dichos sistemas.
3. **Implementación** de controladores empleando la tecnología disponible.

Se suele considerar subcampo de la Ingeniería eléctrica:

- Pero sólo porque muchos controladores son eléctricos.
- En realidad no tiene por qué, también existen controladores mecánicos.
- Incluso hay sistemas *software* controlados por controladores Software.

Sistemas de control (controladores)

Regulan el **comportamiento** de otros sistemas mediante bucles de control.



Sistema de control automático: Diseñado para funcionar sin intervención.

Error y rebasamiento en un controlador

Error: Diferencia entre estado actual y estado deseado de un sistema.

Rebasamiento: Magnitud o dirección cuando el estado supera el *set point*.

Ambos son dos tipos de divergencias. Pueden ofrecer diferente información:

- **Existe/no existe** error: La menor cantidad de información.
- **Dirección**: Hacia dónde hay que ir para minimizar el error.
- **Magnitud**: La distancia al estado objetivo.

Controlar un sistema es mejor cuando conocemos dirección y magnitud.

Clasificación según anticipación a la salida

Punto de vista respecto la relación entre salida y los valores actual y pasados.

Causales: La salida es consecuencia del valor actual y pasado de la entrada.

- Son con los que trabajaremos normalmente porque modelan sistemas reales

No causales: No es posible determinar la salida en función de la entrada.

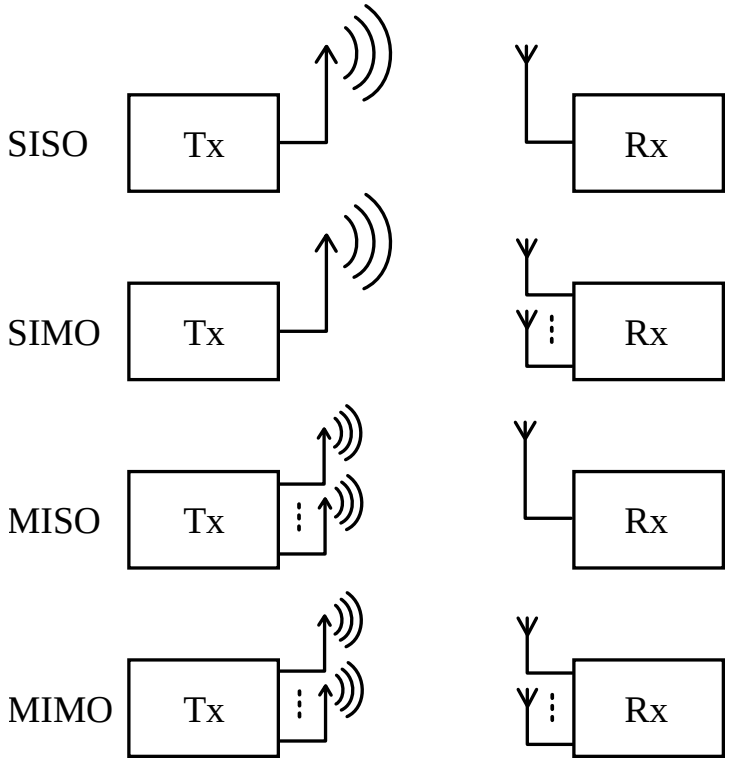
- No existen físicamente, son representaciones abstractas

Estos controladores se diseñan de tal manera que la salida depende de valores futuros de la entrada.

Clasificación según número de entradas y salidas

Clasificación sencilla en función de si hay una o muchas entradas o salidas:

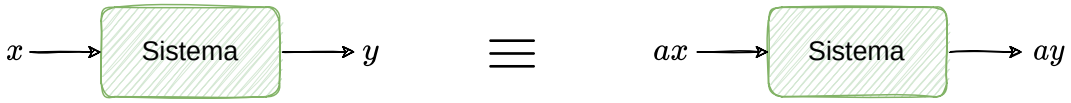
- **SISO** (Single input, single output)
- **SIMO** (Single input, multipe output)
- **MISO** (Multiple input, single output)
- **MIMO** (Multiple input, multiple output)



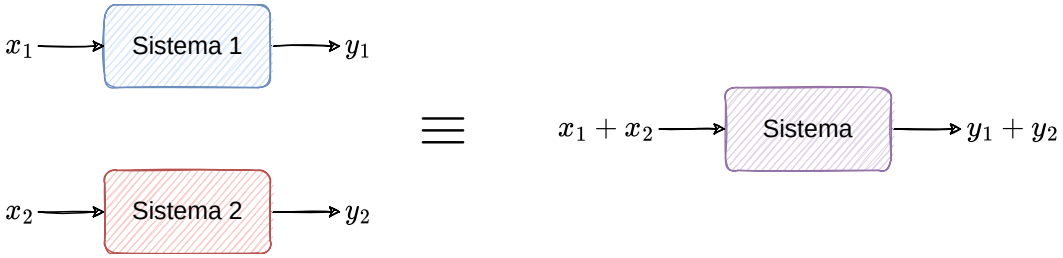
Clasificación según función de transferencia

Un sistema es lineal si su función característica cumple los principios de:

Homogeneidad



Superposición



Por tanto el controlador se denominará:

- **Lineal**: Si cumple ambos principios de superposición y homogeneidad.
- **No lineal**: Si no cumple al menos uno de ellos.

Clasificación según paso del tiempo

Otro punto de vista: ¿cómo se modela el paso del tiempo en un sistema?:

- De **tiempo continuo**: El tiempo evoluciona de manera continua.
- De **tiempo discreto**: El tiempo evoluciona de manera discreta.
- De **eventos discretos**: La tiempo evoluciona cuando ocurren ciertos eventos.

Clasificación según relación entre las variables de entrada

Cuando hablamos de varios controladores, estos se pueden clasificar como:

- **Acoplados**: Si las variables de ambos están relacionadas entre sí.
- **Desacoplados**: Si no lo están.

Clasificación según evolución de parámetros internos

Los controladores mantienen parámetros que modulan su respuesta.

Así diferenciamos dos tipos de controladores:

- **Estacionarios**: Los parámetros no varían durante su funcionamiento.
- **No estacionarios**: Los parámetros pueden variar a lo largo del tiempo.

Clasificación según respuesta del sistema

La salida de un sistema petenece a un dominio, por lo que podemos clasificarlos:

- **Estables**: Para toda entrada acotada la respuesta es acotada.
- **Inestables**: Al menos una entrada acotada produce una salida no acotada.

Clasificación según realimentación

Realimentación: Relación secuencial de causas y efectos entre variables.

- O de otro modo, cuando una o más variables de salida se pasan a la entrada.
- También se la conoce como **retroalimentación** o ***feedback***.
- Concepto muy antiguo, aunque fue formalizado por Norbert Wiener en 1948.

Dependiendo de la acción correctiva que tome el sistema:

- Si es apoyar la salida: Realimentación positiva o "efecto bola de nieve".
- En caso contrario: Realimentación negativa o regulación autocompensatoria.

Dos tipos, de **lazo cerrado** y de **lazo abierto**

Control de lazo cerrado

Cuando se usa la realimentación para minimizar el error de la salida.

- El controlador usa el *feedback* para conocer en cada momento la salida real.

El *feedback* provee al controlador de un comportamiento correctivo:

1. El controlador monitoriza una variable de salida (PV, de *Process Variable*).
2. La compara con la referencia, consigna o punto de ajuste (SP, de *set point*).
3. $SP - PV$ da lugar a la **señal de error**, que es la salida a minimizar

Ejemplos de estos sistemas de control:

- Convergencia fonética de un humano.
- Control de crucero de un vehículo.

Control de lazo abierto

Aquellos controladores que no tienen en cuenta su influencia en el entorno.

Ejemplos de estos sistemas de control:

- Tostadora (las hay que comprueban el color de la rebanada).
- Secadora estándar (las hay que comprueban la humedad del tambor de secado).

Controladores de lazo abierto vs. lazo cerrado

Lazo abierto

Ventajas

- Sencillos, de fácil mantenimiento

Inconvenientes

- Requieren calibración inicial
- Sensibles a perturbaciones
- Mejor en modelos simples

Lazo cerrado

Ventajas

- Control de sistemas inestables
- Robustez frente perturbaciones

Inconvenientes

- Mayor coste (más sensores)
- Son más complejos de modelar

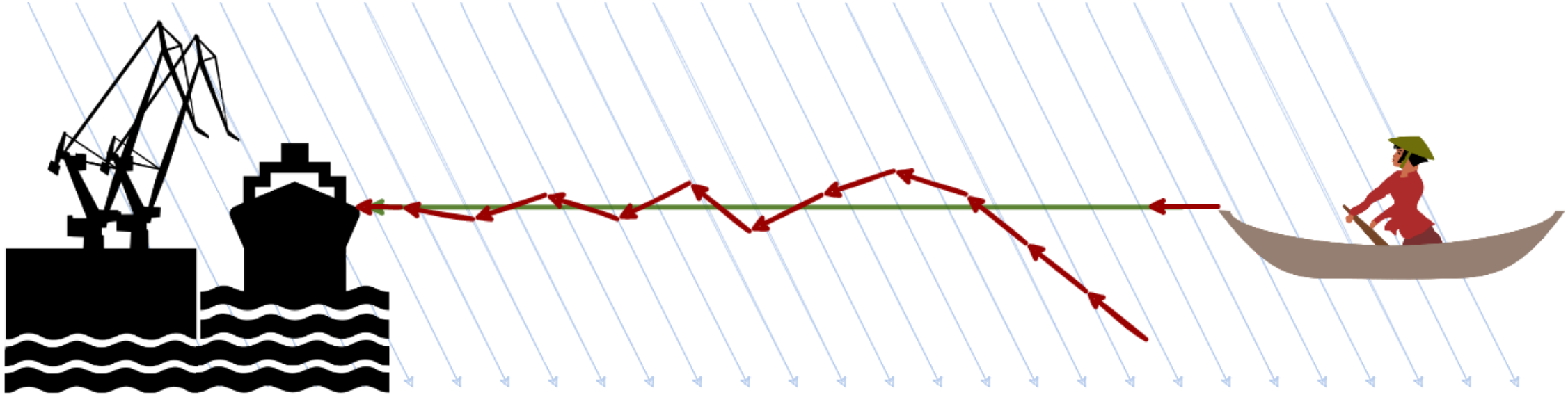
Clasificación según predictibilidad

En función de lo predecible de la respuesta de un sistema, lo podemos clasificar como:

- **Determinista**: Si su comportamiento es extremadamente predecible.
- **Estocástico**: Si es imposible predecir su comportamiento futuro.

Cibernética

Etimología



La palabra timonel (en inglés *steersman*) viene el griego antiguo *kybernetes*:

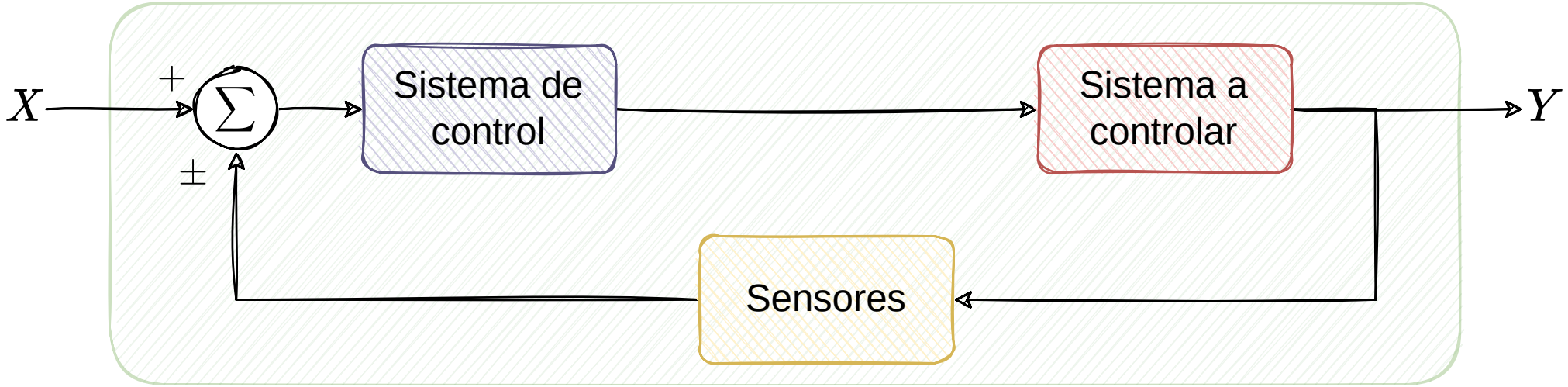
- Los romanos la usaron para su *gubernare* (no eran muy buenos navegando).
- Norbert Wiener tomó la palabra griega y le añadió el sufijo *ics*[3]

Podemos definir la **cibernética** como el **arte de** gobernar o **controlar**.

3 En realidad remplazó el sufijo -tes (actor, agente) por -ike (disciplina, práctica, actividad), pasando de κυβερνήτης (kybernetes) a κυβερνητική (kybernetike). Disculpas por anticipado a todo estudiante de griego clásico.

Sistema de control genérico

Desde el punto de vista de la cibernética, un sistema tiene la siguiente forma:



Generalmente son bucles de control con realimentación

- Realimentación positiva o negativa (de ahí el ± en la generación del input).
- Puede haber sistemas de lazo abierto, pero no suelen ser de interés aquí.

Componentes más importantes de la cibernética

Realimentación: Mejora el rendimiento dinámico del sistema.

- Es un principio muy general que abarca tecnología, astronomía, biología, ...

Información: Flujos de datos que rodean un sistema.

Modelo: Basada en que existe isomorfismo[5] entre diferentes sistemas.

Ley de Ashby[6]: *"cuanto mayor es la variedad de acciones, mayor es la variedad de perturbaciones a controlar"*.

- Sólo podemos controlar cuando sistema y controlador comparten variedad.

4 Fue definida por *Shannon* como la cantidad de incertidumbre eliminada que se describe probabilísticamente.
5 Sistemas mecánicos, electrónicos, etc. se pueden describir mediante las mismas ecuaciones diferenciales.
6 En algunos contextos se conoce como la **Ley de la Variedad Requerida**.

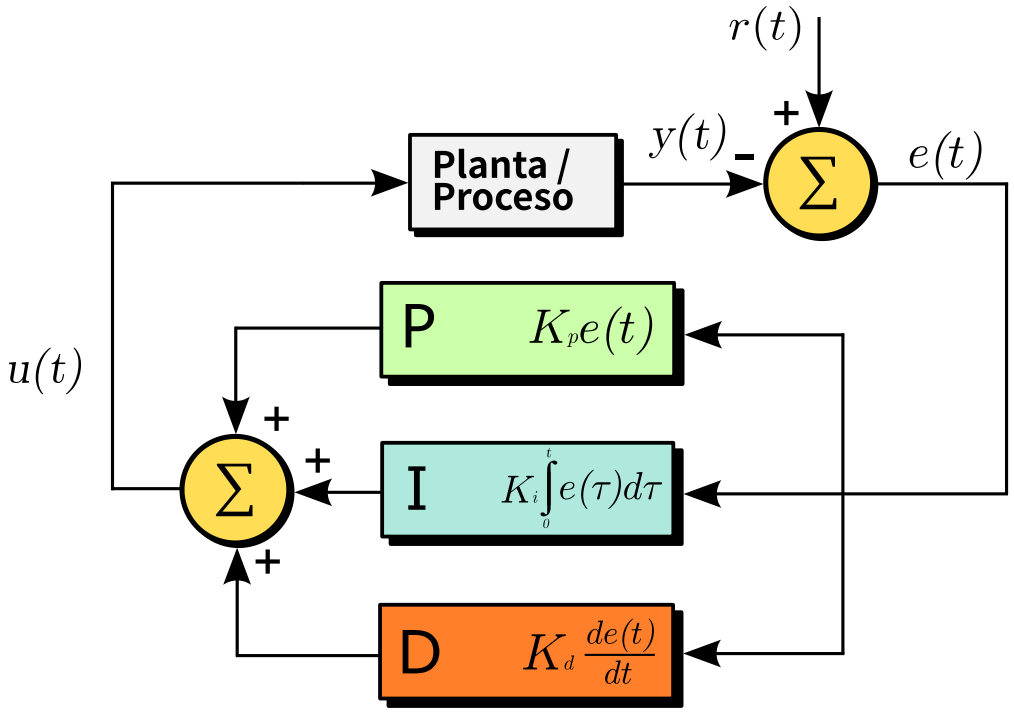
Control clásico

Control clásico con realimentación (principios 1920)

Se basa en la realimentación de la señal de salida al sistema de control.

Modifica la señal de entrada para que la señal de salida se aproxime a la señal deseada, basándose en un modelo matemático del sistema.

Las componentes P (Proportional), I (Integral) y D (Derivative) se pueden combinar para obtener un control más eficiente.



La señal de salida tiene que ser reescalada en función del actuador que se esté utilizando.

Control proporcional P

En el algoritmo de control proporcional, la salida del controlador es proporcional a la señal de error, que es la diferencia entre el punto objetivo que se desea y la variable de proceso:

$$u(t) = K_p e(t)$$

donde K_p es la ganancia proporcional y $e(t)$ es la señal de error.

El control proporcional es el más simple de los controladores, pero también el más inestable. El control proporcional es adecuado para sistemas que tienen un error pequeño y que no cambian rápidamente.

Control integral I

En el algoritmo de control integral, la salida del controlador es proporcional a la integral de la señal de error, que es la suma de las diferencias entre el punto objetivo que se desea y la variable de proceso:

$$u(t) = K_i \int_0^t e(\tau)d\tau$$

donde K_i es la ganancia integral y $e(t)$ es la señal de error.

Aumenta la acción en relación no sólo con el error sino también con el **tiempo** durante el cual ha persistido. Así, si la fuerza aplicada no es suficiente para llevar el error a cero, esta fuerza se incrementará a medida que pase el tiempo.

Control derivativo D

En el algoritmo de control derivativo, la salida del controlador es proporcional a la derivada de la señal de error, que es la velocidad de cambio de la señal de error:

$$u(t) = K_d \frac{de(t)}{dt}$$

donde K_d es la ganancia derivativa y $e(t)$ es la señal de error.

Aumenta la acción en relación no sólo con el error sino también con la **velocidad** a la que cambia el error. Así, si la fuerza aplicada no es suficiente para llevar el error a cero, esta fuerza se incrementará a medida que el error cambie de signo.

Control proporcional-integral-derivativo PID

El control PID es un controlador que combina las tres componentes básicas de control: proporcional, integral y derivativa. El control PID es el más utilizado en la industria.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

donde K_p, K_i y K_d son las ganancias proporcional, integral y derivativa, respectivamente.

Hay que optimizar los parámetros K_p, K_i y K_d para obtener un control eficiente.

Simulador

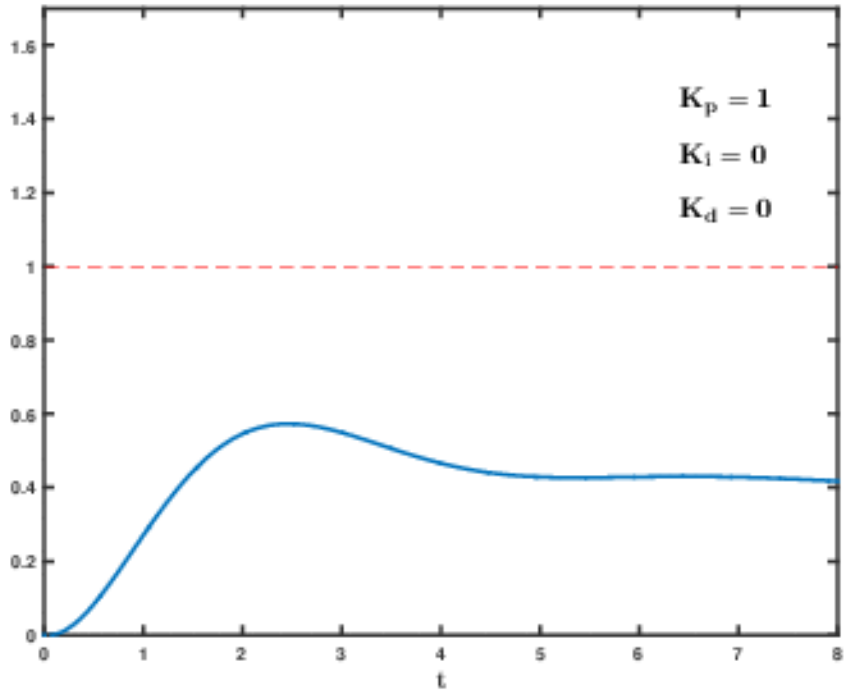
Pseudocódigo del control PID

```
def PID(Kp, Ki, Kd, dt):
    integral = 0
    last_error = 0
    while True:
        error = setpoint - actual_position
        integral += error * dt
        derivative = (error - last_error) / dt
        output = Kp * error + Ki * integral + Kd * derivative
        last_error = error
        yield output
```

`yield` devuelve el valor de la salida del controlador y se suspende hasta la siguiente iteración. Así se define un generador que se puede utilizar en un bucle `for`.

Optimización manual de los parámetros

1. Empezamos con los parámetros a cero.
2. Aumentar K_p hasta que la salida del bucle oscile; entonces, fijar K_p a aproximadamente la mitad de ese valor.
3. Aumentar K_i hasta que cualquier desviación se corrija en tiempo suficiente para el proceso sin causar inestabilidad.
4. Aumentar K_d, si es necesario, hasta que el bucle sea aceptablemente rápido para alcanzar su referencia después de una perturbación.



Optimización Ziegler-Nichols

1. Empezamos con los parámetros a cero.
2. Aumentar K_p hasta obtener una oscilación estable en la salida. Este será el valor K_u y el período de la oscilación será T_u.
   3 Asignar valores a los parámetros K_p, K_i y K_d según la siguiente tabla:

Tipo	K_p	K_i	K_d
PID	$0.6K_u$	$1.2K_u/T_u$	$3K_uT_u/40$
PI	$0.45K_u$	$0.54K_u/T_u$	
P	$0.5K_u$		

Control en cascada

El control en cascada es un método de control en el que se utilizan dos o más controladores para controlar un proceso.

La salida del controlador externo se utiliza como entrada del controlador interno, y representa el punto de referencia del controlador interno.

Se utiliza para controlar procesos que tienen un tiempo de respuesta muy lento, como la temperatura de un horno, una habitación, etc.

Figure 2 – Cascade control system

Fuente de la imagen: Watlow

Ventajas y desventajas

Ventajas:

1. Fácil de implementar (sólo una simple ecuación)
2. Utiliza pocos recursos
3. Resistente a los desajustes de optimización
4. Fácil de optimizar
5. Buena respuesta a las perturbaciones

Desventajas:

1. Bajo rendimiento en procesos con largos tiempos de espera
2. Bajo rendimiento para tratar fuertes no linealidades
3. Dificultad para manejar múltiples variables con fuerte interrelación
4. Dificultad para manejar múltiples restricciones

Control borroso

Recordatorio de lógica borrosa

Se puede considerar una extensión de la teoría clasica de conjuntos:

- En esta teoría, los elementos pertenecen o no a un conjunto
- Función característica: $f(x) = 1$ si $x \in A$ y $f(x) = 0$ si $x \notin A$

Trata información a priori imprecisa en términos de conjuntos borrosos:

- Los elementos pertenecen a un conjunto con un grado de pertenencia.
- Función de pertenencia: $f(x) = \mu(x) \in [0, 1]$

Los conjuntos borrosos se agrupan en particiones

- Una partición se define sobre una variable denominada lingüística.

Definiciones

Variable lingüística: Variable cuyos valores son términos en lenguaje natural.

Partición borrosa: Todos los conjuntos borrosos de una variable lingüística.

Función de pertenencia: Determina el grado de pertenencia de un elemento a un conjunto borroso (en tanto por uno).

Ejemplo: La variable lingüística $precio$ *puede tomar los valores* $precio \equiv \{barato, normal, caro\}$*. Estos serán tres conjuntos borrosos, cada uno con las funciones de pertenencia* $\{f_{barato}(x),\ f_{normal}(x)$ *y* $f_{caro}(x)\}$*.*

Operaciones borrosas

Complemento: $f'_{barato}(x) = 1 - f_{barato}(x)$

t-normas (intersección)

- Mínimo: $f_{barato} \cap f_{normal} = min(f_{barato}, f_{normal})$
- Producto algebráico: $f_{barato} \cap f_{normal} = f_{barato} \cdot f_{normal}$

t-conormas (unión)

- Máximo: $f_{barato} \cup f_{normal} = max(f_{barato}, f_{normal})$
- Suma algebráica: $f_{barato} \cup f_{normal} = f_{barato} + f_{normal} - f_{barato} \cdot f_{normal}$

La **inferencia** ($\rightarrow$) se suele definir como la operación de **intersección**.

Reglas borrosas

Son reglas que relacionan varios antecedentes con consecuentes, donde:

- Antecedentes: Conjuntos borrosos de entrada
- Consecuentes: Conjuntos borrosos de salida

***Si** el precio es barato **Y** la calidad es mala **entonces** la satisfacción es baja.*

Se agrupan en una **base de reglas**, las cuales pueden ser de varios tipos:

- De tipo Mandani: **Si** V_1 es $F_i^{V_1}$ **Y** V_2 es $F_j^{V_2}$ **Y . . . entonces** V_o es $F_k^{V_o}$
- De tipo Sugeno: **Si** V_1 es $F_i^{V_1}$ **Y** V_2 es $F_j^{V_2}$ **Y . . . entonces** $V_o = f(\vec{x})$

Fuzzification y defuzzification

Fuzzification: Convertir valores de entrada concretos en conjuntos borrosos.

- Es basicamente aplicar las funciones de pertenencia a los valores de entrada.

Defuzzification: Convertir conjuntos borrosos en valores de salida concretos.

- Existen muchas técnicas para realizar esta operación.
- Las más comunes son el centroide y el centroide simplificado

Centroide

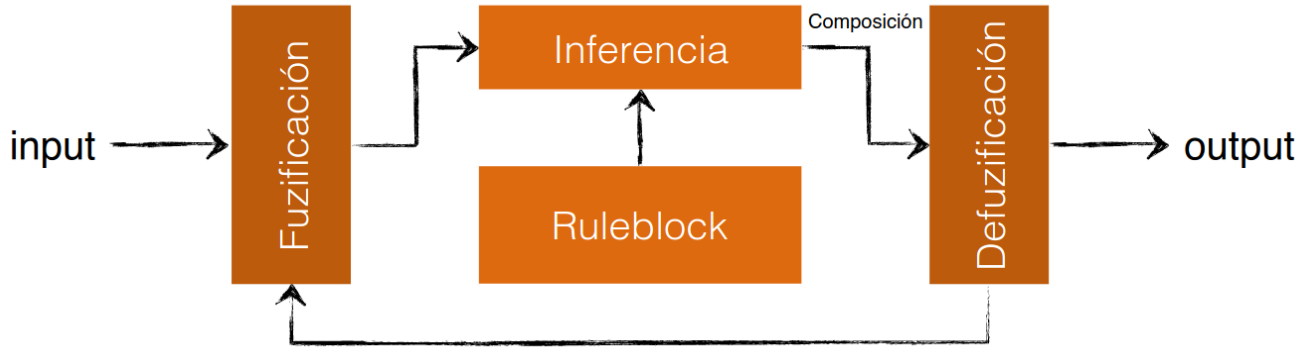
$$y = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy}$$

Centroide simplificado

$$y \approx \frac{\sum y \cdot \mu(y)}{\sum \mu(y)}$$

Controlador borroso

Es un sistema de control que se apoya en la lógica borrosa como sigue:



1. Toma la entrada al sistema.
2. Pasa los valores a pertenencia a conjuntos borrosos (*fuzzification*)
3. Infiere conjuntos de borrosos de salida haciendo uso de las reglas borrosas.
4. Pasa los conjuntos borrosos de salida en valores concretos (*defuzzification*)
5. Aplica la salida al sistema a controlar.

Ejemplo de implementación de un controlador borroso

Diseño de un controlador borroso

Para diseñar un controlador borroso, se debe seguir el siguiente proceso:

1. Identificar variables de entrada y de salida.
2. Determinar los conjuntos borrosos para cada variable
3. Definir las reglas borrosas que van a regir el comportamiento del controlador.
4. (Opcional) Normalización y escalado de entradas y salidas.

Implementaremos un controlador para el *problema de las propinas:*

- Problema clasico de control borroso.
- ¿Cuánto dar de propina en función de la calidad del servicio y de la comida?
- Usaremos la biblioteca `skfuzzy` para implementar un controlador borroso.

Formulación del problema

Antecedentes (entradas):

- Servicio (de 0 a 10): $malo$, $normal$, $bueno$
- Calidad (de 0 a 10): $mala$, $aceptable$, $buena$

Consecuentes (salidas):

- Propina (de 0 a 25): $baja$, $media$, $alta$

Reglas:

1. **Si** Servicio $bueno$ o Calidad $buena$ **entonces** Propina $alta$
2. **Si** Servicio $normal$ **entonces** Propina $media$
3. **Si** Servicio $malo$ y Calidad $mala$ **entonces** Propina $baja$.

Implementación de las variables lingüísticas

El primer paso es definir las variables de entrada y salida del controlador.

```
import numpy as np
from skfuzzy import control as ctrl

# Antecedentes
servicio = ctrl.Antecedent(np.arange(0, 11, 1), 'servicio')
calidad = ctrl.Antecedent(np.arange(0, 11, 1), 'calidad')
# Consecuente
propina = ctrl.Consequent(np.arange(0, 26, 1), 'propina')
```

Definición de los conjuntos borrosos

Para cada variable, se definen los conjuntos borrosos que la componen.

```
import skfuzzy as fuzz

# Conjuntos borrosos de servicio
servicio['malo'] = fuzz.trimf(servicio.universe, [0, 0, 5])
servicio['normal'] = fuzz.trimf(servicio.universe, [0, 5, 10])
servicio['bueno'] = fuzz.trimf(servicio.universe, [5, 10, 10])
# Conjuntos borrosos de calidad
calidad['mala'] = fuzz.trimf(calidad.universe, [0, 0, 5])
calidad['aceptable'] = fuzz.trimf(calidad.universe, [0, 5, 10])
calidad['buena'] = fuzz.trimf(calidad.universe, [5, 10, 10])
# Conjuntos borrosos de propina
propina['baja'] = fuzz.trimf(propina.universe, [0, 0, 13])
propina['media'] = fuzz.trimf(propina.universe, [0, 13, 25])
propina['alta'] = fuzz.trimf(propina.universe, [13, 25, 25])
```

Se puede usar el método `.automf(n)` para definirlos de forma automática.

Visualización de los conjuntos borrosos

Para visualizar los conjuntos borrosos, se puede usar la función `view()`.

```
servicio.view()
calidad.view()
propina.view()
```

Concretamente mostrará la variable lingüística junto con:

- Las **funciones de pertenencia** que caracterizarán a cada conjunto borroso.
- El **dominio** de la variable lingüística.

Definición de las reglas

Para definir las reglas, se debe usar la función `ctrl.Rule()`.

```
rulebase = [
    ctrl.Rule(servicio['bueno'] | calidad['buena'], propina['alta']),
    ctrl.Rule(servicio['normal'], propina['media']),
    ctrl.Rule(servicio['malo'] & calidad['mala'], propina['baja'])
]
```

Suele ser buena costumbre definir las reglas en una lista.

Definición del controlador

Para definir el controlador, se debe usar la función `ctrl.ControlSystem()`.

```
>>> controlador = ctrl.ControlSystem(rulebase)
```

Luego se simula con la función `ctrl.ControlSystemSimulation()`.

- Este objeto se encarga de implementar casos concretos sobre un controlador.

```
>>> simulacion = ctrl.ControlSystemSimulation(controlador)
```

- El caso concreto se simulará con la función `compute()`.

```
>>> simulacion.input['calidad'] = 6.5
>>> simulacion.input['servicio'] = 9.8
>>> simulacion.compute()
>>> print(simulacion.output['propina'])
19.847607361963192
```

Razonamiento en robots

Inteligencia

¿Qué es la inteligencia?

- ¿Sumar y restar números grandes? ¿Resolver una ecuación diferencial?
- ¿Saber jugar al GO? ¿Ganar al GO?
- ¿Reconocer a una persona? ¿A un gato?
- ¿Conducir un coche? ¿Una moto? ¿Un avión?
- ¿Ser capaz de andar por la calle sin tropezar con mucha gente alrededor?
- ¿Entender lo que dice una persona? ¿Dobles sentidos? ¿Ironía?

¿Puede una máquina ser inteligente?

¿Somos algo más que datos, reglas y cálculos?

Basic Questions (John McCarthy) - <www-formal.stanford.edu/jmc/whatsai/node1.htm>

Algunas definiciones

	Inteligencia humana	Ideal de inteligencia
Razonamiento	Estudio de **procesos** que posibilitan **razonar** y **actuar**.[1]	"**Máquinas con mente**", en un sentido literal.[2]
Conducta	Estudio para que un **ordenador haga cosas que la gente hace mejor**.[3]	**Automatización de la conducta inteligente**.[4]

- **IA Débil**: Aspectos de comportamiento considerados inteligentes.
- **IA Fuerte**: Un agente artificial puede llegar a sentir y tener mente.

[1] Wiston, 1992, [2] Haugeland, 1985, [3] Rich and Knight, 1991, [4] Luger y Stubblefield, 1989

Elementos relacionados con la Inteligencia

La inteligencia es un concepto que se relaciona directamente con:

- **Conciencia**: Tener experiencia subjetiva y pensamiento.
- **Conciencia de sí mismo**: stener experiencia de de los propios pensamientos y del individuo como algo separado.
- **Sentiencia**: Capacidad de sentir percepciones de forma subjetiva.
- **Sapiencia**: Capacidad de sabiduría.

Existen dos debates destacados desde un plano ético respecto a estos:

1. ¿Son necesarios y/o suficientes para considerar un ente inteligente?
2. ¿Sirven de base para otorgar derechos y deberes a un ente?

Razonamiento en robots

Razonamiento: Proceso de inferencia que permite a un agente obtener conocimiento a partir de información previamente adquirida.

En un robot autónomo **el razonamiento lo provee el controlador**.

- Controlador $\approx$ cerebro de un robot.

Un robot autónomo intenta alcanzar varios objetivos a la vez:

- Comportamientos simples de supervivencia (e.g. no quedarse sin energía)
- Actividades complejas (e.g. jugar al fútbol).

Balance tiempo/reacción en el control de robots

Una reacción debe ser rápida, mientras que el pensamiento es lento.

- Pensar permite planificar cómo evitar situaciones peligrosas o desfavorables.
- Lo malo, pensar mucho puede ser peligroso (e.g. caer en una zanja).

Para "pensar", un robot requiere de mucha información de entornos complejos.

- Son necesarios modelos para representar el entorno del robot

Niveles de complejidad cognitiva que controlar

Control a **bajo nivel**: Tareas simples (e.g. cambiar de marcha en un vehículo).

- Generalmente se tratan de problemas de valores continuos.
- Escalas temporales cortas, de frecuencas mayores a 1 Hz.

Control a **nivel intermedio**: Tareas medias (e.g. cambiar de carril).

- Involucran indistintamente tareas de valor continuo o discreto.
- Escalas temporales medias, del orden de unos pocos segundos

Control a **alto nivel**: Tareas complejas (e.g. planificar una ruta completa).

- Normalmente son problemas de valores discretos.
- Escalas de tiempo grandes, incluso de minutos.

Arquitecturas de control

Arquitecturas de control en robots

Un robot es un tipo de aplicación muy diferente a otras:

- Sus objetivos de funcionamiento suelen ser más complejos.
- Sus entornos suelen ser dinámicos y no estructurados.

Una arquitectura de control define los principios de diseño de un robot:

- Estructura arquitectónica: Subsistemas que lo componen y cómo interactuan.
- Estilo arquitectónico: Conceptos computacionales subyacentes en el sistema.

Dos tipos bien diferenciados: Arquitecturas **deliberativas** y **reactivas**.

Estructura y estilo arquitectónico

Todos los robots comparten las mismas primitivas: percibir, planificar y actuar.

- **Percibir**: Obtener información del entorno a través de sus sensores.
- **Planificar**: Determinar acciones a realizar segun estados de entorno y robot.
- **Actuar**: Realizar las acciones planificadas.

Los diferentes estilos arquitectónicos surgen a partir de:

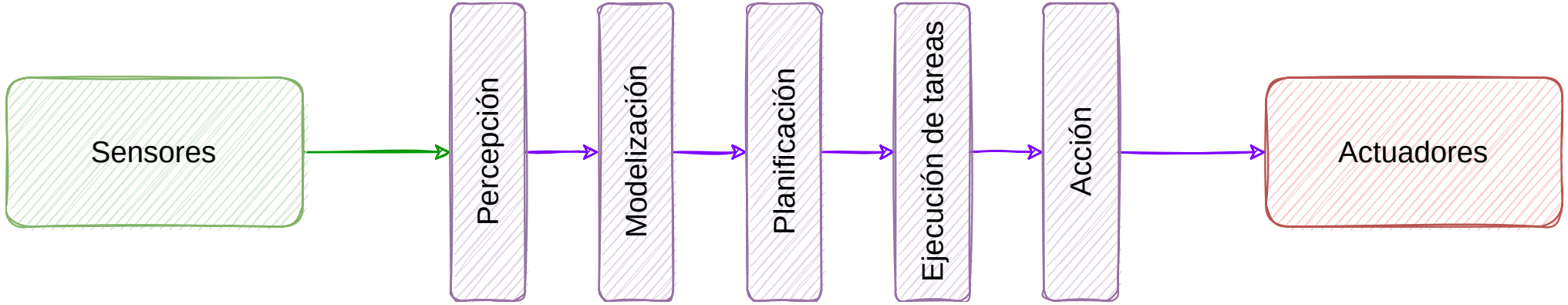
- ¿Cómo se relacionan las primitivas?
- ¿Dónde se toman las decisiones correspondientes a la planificación?
- ¿De qué manera se procesa y distribuye la información sensorial?

La estructura se relaciona a la implementación concreta del estilo.

Arquitectura deliberativa (o jerárquica)

Las primitivas se relacionan de manera secuencial:

1. Se percibe el entorno y se construye un modelo de este y del estado actual.
2. Se planifica la acción o acciones para acercar al robot a su objetivo.
3. Se ejecutan las acciones planificadas.



Es un punto de vista *top-down* de la concepción de un robot.

Fue el primer paradigma planteado y el más usado hasta principio de los 90.

- Útil para robots sencillos, pero desafiante según aumentaba su complejidad.

La información necesita atravesar todos los módulos de la arquitectura:

- Cualquier fallo en algún componente provoca un fallo total del sistema.
- Entorno y estado son cambiantes $\rightarrow$ Problema de mantenimiento de modelo.
- También problemático el rectificar ante información errónea o incompleta.

Y además, este paradigma se basa en la hipótesis de mundo cerrado[5]:

- Suposición no asumible en prácticamente ningún robot.

[5] Supuesto en el que se asume que la información obtenida por el sistema es siempre completa, es decir, conoce todo lo que ocurre en su entorno y por tanto no se va a encontrar con situaciones inesperadas.

Principales inconvenientes de las arquitecturas deliberativas

Estas arquitecturas tienen varios inconvenientes, entre los que se encuentran:

- **Modelizar** el entorno es muy **costoso** en términos de **tiempo** y **memoria**.
- La **planificación** también suele ser **lenta** y requerir mucha **memoria**.
- La **planificación no lineal** es **intratable** (es un problema NP-completo).
- La **retroalimentación** a través del modelo del entorno es **complicada**.
- Una única línea entre una detección y su actuación.
- Enfoque muy general, **pobre** para muchas **tareas específicas**.
- **Pasar representaciones** entre diferentes componentes es **costoso**.

Una pausa para reflexionar

¿Cómo es la arquitectura de control de una mosca?

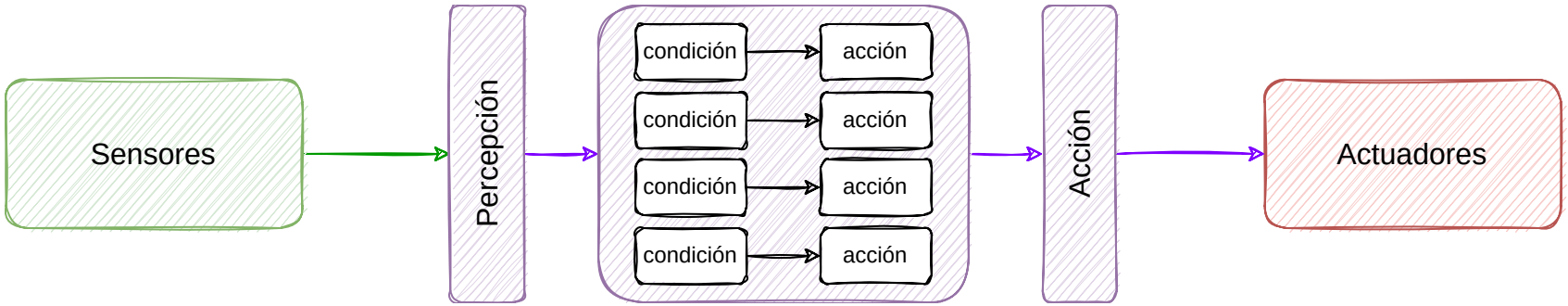
- Crea un modelo del entorno por el que navega.
- Se plantea la naturaleza de las amenazas.
- Analiza la idoneidad de plantar huevos en heces.
- Delibera sobre cómo aterrizar en superficies irregulares.
- Sensores y actuadores están estrechamente conectados.
- Patrones de comportamiento aprendidos, no planificados.
- Técnicas de navegación sencillas (casi deterministas).
- Miles de receptores visuales simples conectados al cerebro.

Para navegar por un entorno, ¿quién lo hace mejor, una mosca o un dron?

Arquitectura reactiva

Surgió como una respuesta a los problemas de las arquitecturas deliberativas.

- Se basa en la idea de que el robot no necesita planificar acciones.
- Más orientado hacia la biología y psicología de los seres vivos.
- No hay animales de propósito general, así que ¿por qué robots sí?



Se elimina completamente la fase de planificación.

- La percepción provoca directamente una respuesta en la acción.

Es un punto de vista *bottom-up* de la concepción de un robot.

- Existencia de un flujo único entre percepción y acción para cada comportamiento.
- Los **mapas de estímulo-respuesta** los hacen **intrínsecamente paralelos**.

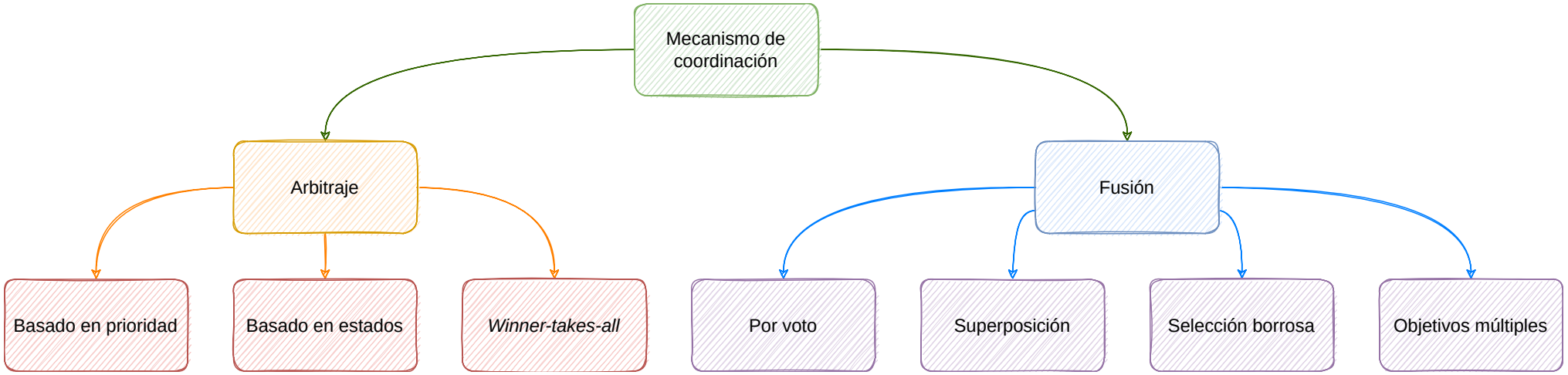
Este paradigma ofrece muchas ventajas sobre el deliberativo, entre ellas:

- **No requieren** de **modelo de entorno** ni de **planificación** de ningún tipo.
- Sí necesitan **mecanismos** de **retroalimentación**, a poder ser **cortos**.
- **Muy específico**, es decir, bueno en una o dos tareas conretas.
- **No** se pasan **representaciones entre componentes**.
- Mayor **resiliencia** por la (teórica) independencia de los comportamientos.
- Mayor **facilidad de diseño** de cada módulo independiente.

Coordinación de comportamientos

Varios comportamientos se pueden coordinar en dos estrategias principales:

- **Competitiva** (arbitraje): Sólo se selecciona la salida de un comportamiento.
- **Cooperativa** (fusión): Se combinan las salidas de varios comportamientos.



Estas salidas resultado se suelen denominar comportamiento emergente[6].

6 Comportamiento complejo surgido de la coordinación de comportamientos más simples (e.g. colonias de hormigas).

Arquitectura basada en comportamientos (BBR)

Tipo de arquitectura reactiva que utiliza sistemas biológicos como modelo:

- **Adaptabilidad**: No se depende de cálculos preestablecidos.
- **No necesitan modelar entorno**, toda la información se obtiene de los sensores.
- Esa información se usa para **corregir gradualmente sus acciones**.

Esas arquitecturas muestran acciones en apariencia más biológica.

- De hecho las comparaciones entre BBR e insectos son muy frecuentes.
- Algunos investigadores lo consideran un **ejemplo de IA débil**.

Arquitecturas híbridas

Aprovecha las fortalezas de ambos paradigmas suavizando sus debilidades.

- Suele ser la opción más elegida en la actualidad.

En estas arquitecturas suelen existir tres capas básicas:

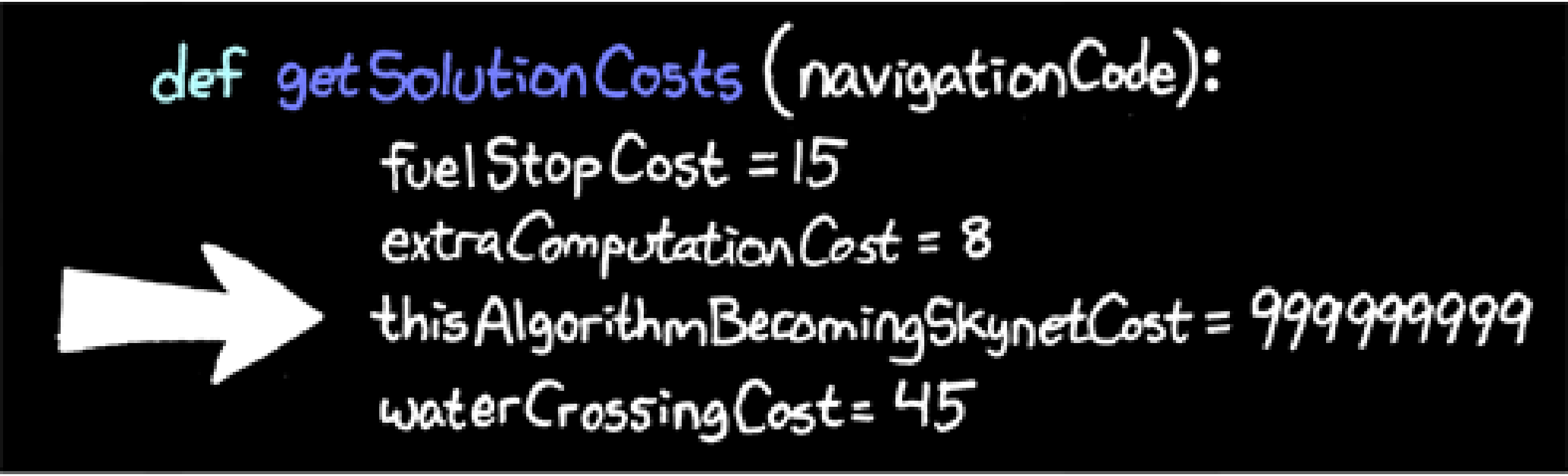
- Capa reactiva de bajo nivel, relacionada con tareas cognitivas de bajo nivel.
- Capa deliberativa de alto nivel, relacionada con tareas cognitivas de alto nivel.
- Capa intermedia, que actúa como puente entre las dos capas anteriores.

Se suelen descomponer, a su vez, en más o menos componentes.

- No hay una solución global, la estructura suele depender mucho del problema.

Optimización

Computación evolutiva para la optimzación de controladores



GENETIC ALGORITHMS TIP:

ALWAYS INCLUDE THIS IN YOUR FITNESS FUNCTION

Comentar conceptos de:

- Espacio de estados.

Que vienen bien aquí y también en entrenamiento por refuerzo.

COSAS DE LA QUE HABLAR

Equilibrio entre exploración y explotación

Exploración: Explorar es espacio de estados para localizar regiones prometedoras para soluciones al problema.

Explotación: Aprovechar la información conocida para centrar la búsqueda en regiones prometedoras concretas.

Aprendizaje por refuerzo para optimizar comportamientos

Paradigmas de aprendizaje en *Machine Learning*

Supervisado: Se aprende de ejemplos con sus correspondientes respuestas.

- Problemas de regresión y clasificación.

No supervisado: Búsqueda de patrones en datos no etiquetados.

- Problemas de *clustering*, reducción de la dimensionalidad, recodificación, ...

Por refuerzo: Se aprende a través de la experiencia a base de recompensas.

- Problemas de aprendizaje de políticas de decisión.
- No se le presentan ejemplos-respuestas
- La evaluación del sistema es concurrente con el aprendizaje.

"Las respuestas que producen un efecto positivo en una situación concreta aumentan la probabilidad de repetirse en dicha situación, mientras que las que producen un efecto negativo la reducen."

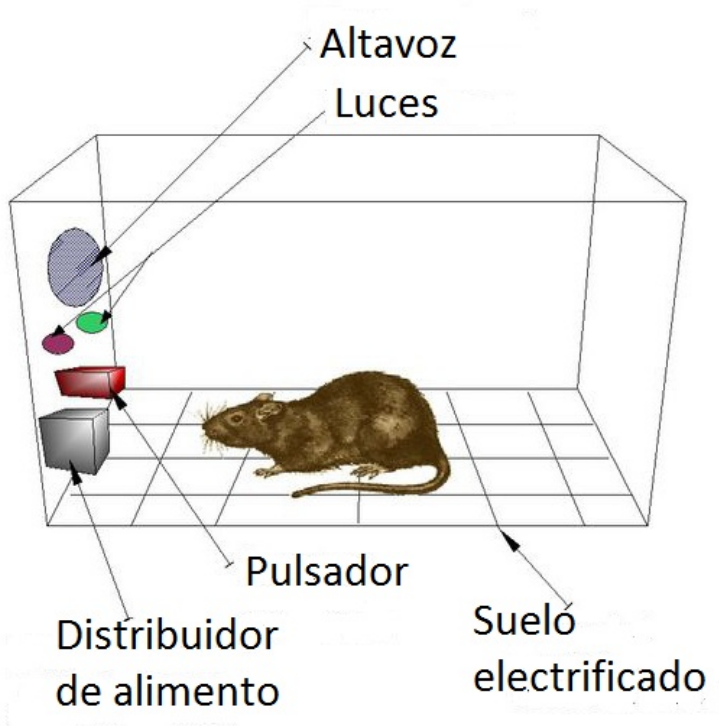
- Edward Thorndike - Law of Effect (1898) -

Caja de Skinner

Experimento desarrollado en 1938 por Burrus F. Skinner.

- También **cámara del condicionamiento operante**.
- ¿Animal realiza acción deseada? Recompensa
- ¿No? Penalización

Se vio que algunos comportamientos de aprendizaje son bucles observación-acción-recompensa



Aprendizaje por refuerzo (RL)

Área del *machine learning* donde **los agentes aprenden interactuando**:

- **Imita** de manera fundamental el **aprendizaje** de muchos **seres vivos**.
- Esa interacción produce tanto resultados deseados como no deseados.
- Se entrena con la **recompensa o castigo** determinados para dicho resultado.
- El agente tratará de maximizar la recompensa a largo plazo.

Se utiliza principalmente en dos áreas hoy en día:

- **Juegos**: Los agentes aprenden las reglas y las jugadas jugando[1].
- **Control**: Los agentes aprenden en entornos de simulación las mejores políticas de control para un problema determinado.

[1] Un ejemplo curioso es el publicado en https://www.nature.com/articles/nature14236, donde describen cómo un agente aprende a jugar a 49 juegos de Atari 2600 llegando a un nivel de destreza comparable al humano.

Terminología

Agente inteligente (agente, robot): Entidad que interactúa con el **entorno**.

Espacio de **estados** S y de **observaciones** O: Información obtenida del entorno:

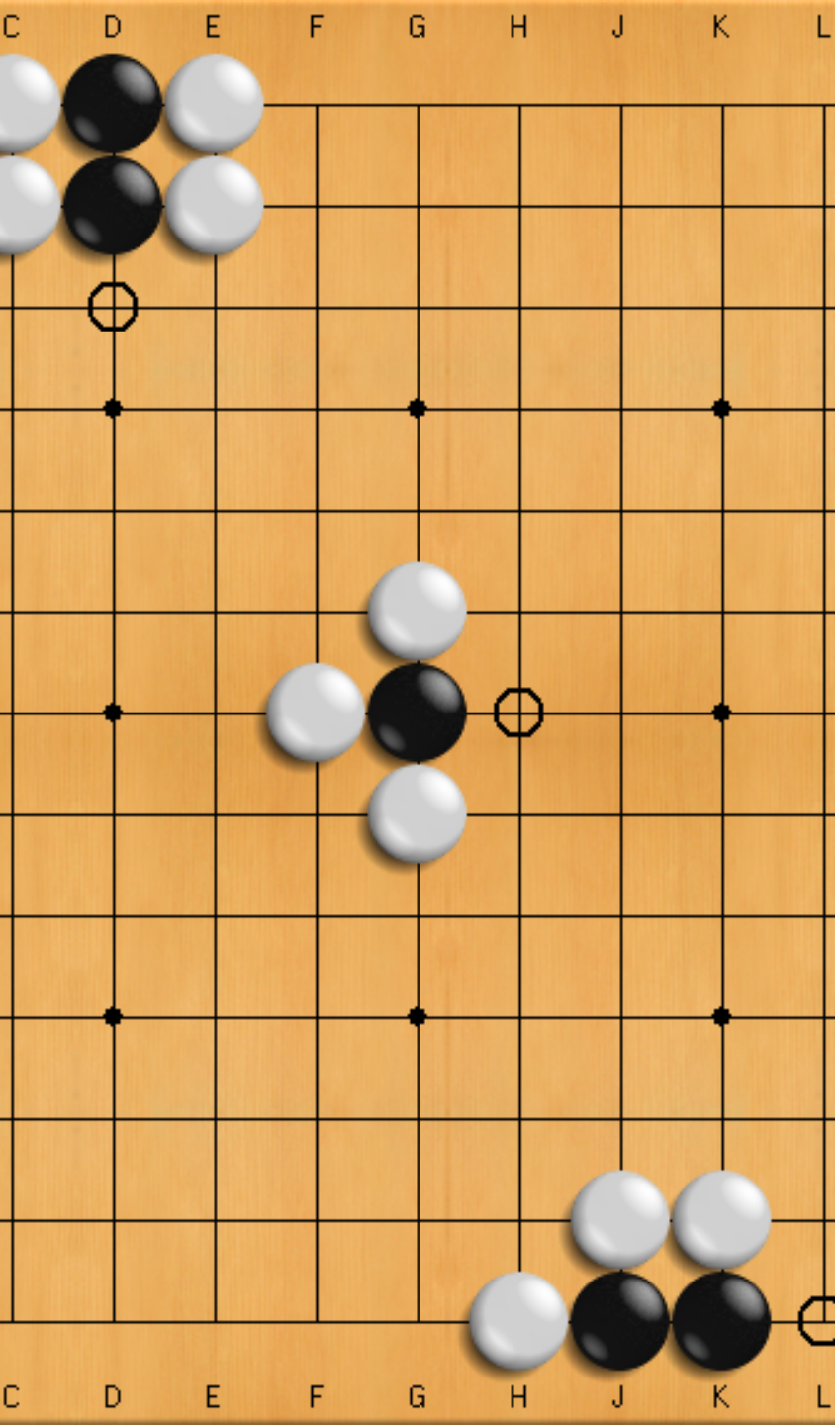
- **Estado** $s_t \in S$: Descripción **completa** del estado del entorno en un instante t.
- **Observación** $o_t \in O$: Descripción **parcial** del estado del entorno en un instante t.

Espacio de acciones A: Conjunto de acciones que puede realizar el agente:

- **Discreto**: El conjunto es finito (e.g. juego del Go).
- **Continuo**: El conjunto es infinito (e.g. vehículo autónomo).

Conjunto de recompensas R: Todas las recompensas que puede recibir un agente.

- $r_t \in R$: La recompensa recibida por el agente en un instante t.



Ejemplo #1: Juego del Go

- Agente: Robot que juega al Go.
- Entorno/mundo: El tablero en el que se juega.
- Estado: Colocacion concreta de las piedras.
- Observación: Estado (sin información oculta).
- Espacio de acciones (finito): Poner piedra en una casilla vacía.

Ejemplo #2: Warcraft II

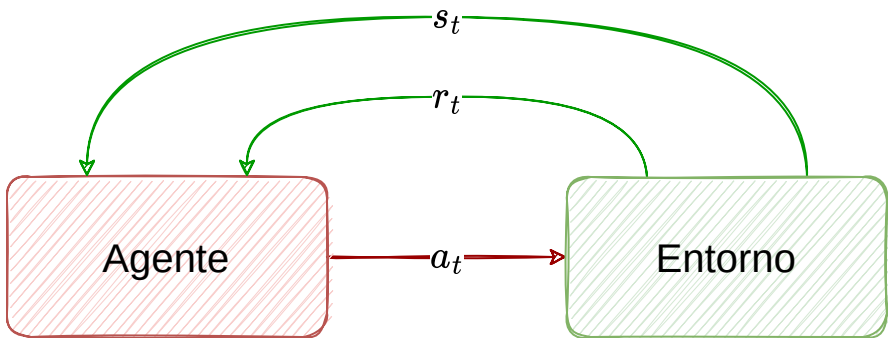
- Agente: Robot que juega al Warcraft II.
- Entorno/mundo: Pantalla en la que se juega.
- Estado: Situación de la pantalla en un momento determinado.
- Observación: Lo que el agente ve en un instante determinado (sin la niebla de guerra).
- Espacio de acciones (finito): Mover unidades, construir edificios, ...

Ejemplo #3: Coche autónomo

- Agente: Robot que conduce el vehículo.
- Entorno/mundo: El continente en el que se encuentra el vehículo.
- Estado: Estado del continente en un momento determinado.
- Observación: Lo que el agente ve por sus sensores en un instante determinado.
- Espacio de acciones (infinito): Girar el volante un determinado ángulo, aumentar y disminuir aceleración, ...

Modelo de interacción agente-entorno

El proceso de aprendizaje por refuerzo es el siguiente:



1. El agente lee un estado s_0 del entorno.
2. De acuerdo a s_0, realiza la acción a_0.
3. El entorno pasa al nuevo estado s_1.
4. El agente recibe una recompensa r_1.
5. Iterar hasta encontrar estrategia óptima

Este bucle produce una secuencia de estados, acciones y recompensas:

$$s_0, a_0, r_1, s_1, a_1, \ldots$$

Markov Decision Processes (MDP)

Propiedad de Márkov

El estado futuro de un proceso depende del estado actual, y no de los anteriores.

- Es un estado que cumplen ciertos procesos estocásticos.
- Definida por Andréi Markov en 1906 en su Teoría de Cadenas de Márkov[2].

Al proceso que satisface esta propiedad se denomina **Proceso de Márkov**.

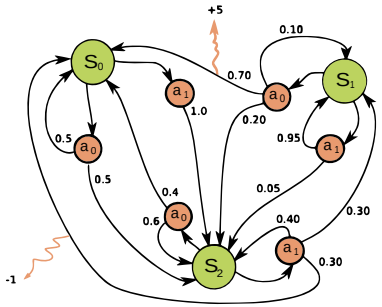
- Concretamente se denominan Procesos de Márkov de **primer orden**.
- La definición se puede extender a n estados anteriores (proceso de orden n).

Si hay que quedarse con algo, nos dice que nuestro agente sólo necesita el estado actual para decidir qué acción tomar.

[2] Más información en https://en.wikipedia.org/wiki/Markov_chain.

Procesos de decisión de Márkov (MDP)

Proceso **estocástico** de **tiempo discreto** que satisface la **propiedad de Márkov**.



Matemáticamente se define como una 4-tupla (S, A, P_a, R_a) donde:

- S y A: Espacios de estados y de acciones del proceso respectivamente.
- $P_a(s, s')$: Probabilidad de que la acción a nos lleve de s a s'.
- $R_a(s, s')$: Recompensa inmediata por pasar del estado s al estado s' con la acción a.

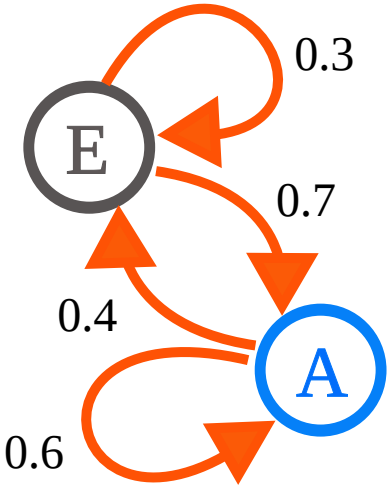
A la función $\pi : S \rightarrow A$ que define las políticas de decisión se le denomina ***policy***.

Diferencia entre un MDP y Cadena de Márkov

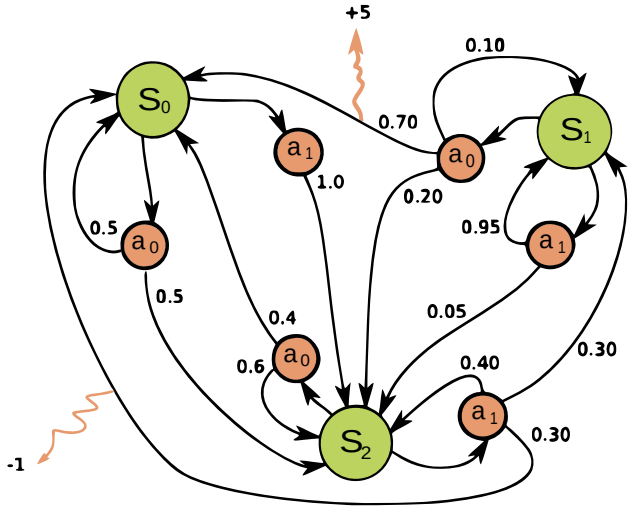
Los MDP extienden a las cadenas de Márkov en dos aspectos:

- Permiten elegir **acciones** para realizar transiciones entre estados.
- Incluyen **recompensas** a una o más de esas transiciones.

Cadenas de Márkov



MDP



Tareas y problemas en aprendizaje por refuerzo

Se entiende por tarea a una instancia de un problema.

Tenemos dos tipos bien diferenciados de tareas:

- **Episódicas**: Poseen estado inicial y terminal o final (e.g. Sonic the Hedgehog).
- **Continuas**: Tarea que no posee estado terminal (e.g. vehículo autónomo).

Es importante de cara a las simulaciones para entrenar a los agentes:

- Una tarea episódica se puede reanudar cuando llega a un estado final.
- Una tarea continua no acaba nunca y es necesario decidir cuando se reinicia.

Recompensas y tomas de decisiones

Hipótesis de la recompensa

El agente quiere **maximizar la recompensa acumulada** (rendimiento esperado).

- Recompensa: *Feedback* que recibe el agente para saber si la acción es buena o no.

Recompensa acumulada: Suma de todas las recompensas de la secuencia.

$$R(\tau) = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

Sin embargo, las recompensas no tienen por qué tener todo su valor siempre.

- De ahí el **factor de ajuste** $\gamma \in [0, 1]$ que se le aplica a la recompensa.
- Las recompensas a corto plazo tienen más probabilidades de suceder.
- γ indica si interesan más recompensas a **corto** ($\gamma \approx 0$)) o a **largo** ($\gamma \approx 1$) **plazo**.

Función de politicas de decisión

La función de *policy* (π) es la que **asigna** una **acción** $a \in A$ a cada **estado** $s \in S$.

- Realiza el mapeo entre el espacio de estados y el de acciones.
- Define completamente el comportamiento de un agente.

Buscamos π que **maximice el rendimiento esperado**; existen dos métodos:

- **Directo**: ¿Qué acción debe realizar en el estado actual?
- **Indirecto**: ¿Qué estados son mejores para tomar la acción que lleva a esos estados?

Métodos directos (*policy learning*)

En estos métodos intentamos **aprender directamente la función π**.

Determinista

Devuelve **siempre la misma acción** para un estado determinado.

$$\pi(S) = A$$

Por ejemplo:

$$\pi(s_t) = \{\blacktriangleright\}$$

No determinista

Devuelve una **distribución de probabilidad** sobre las acciones.

$$\pi(S) = P[A|S]$$

Por ejemplo:

$$\pi(s_i) = \{(\blacktriangleleft, 0.3), (\blacktriangleright, 0.5), (\blacktriangledown, 0.1), (\blacktriangle, 0.1)\}$$

Para aprenderlas se suelen usar redes neuronales (no se verá en esta asignatura).

Métodos indirectos (basados en valores)

Aprendemos una función v_π (o q_π) que **relaciona un estado con su valor estimado**.

- Valor: Recompensa acumulada si empieza en ese estado y se mueve al mejor estado.
- El agente selecciona la acción de mayor valor.

Valor estado

$$v_\pi(s_t) = E_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$$

Valor par estado-acción

$$q_\pi(s_t, a_t) = E_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})]$$

Independientemente de la función elegida, el resultado será la recompensa esperada.

Por cierto, ¿cómo sabemos **qué acciones futuras son óptimas**?

- Spoiler: **No lo sabemos**, actuamos con lo que sabemos en cada momento.

Estrategia ϵ-greedy

Política sencilla para elegir acción que mantiene el equilibrio exploración/explotación.

- Una política basada en la aleatoriedad no da buenos resultados.
- Pero una basada en escoger siempre la mejor opción se estanca en mínimos locales.

La estrategia ϵ-greedy es una **combinación de ambas**.

- Con probabilidad ϵ se escoge una acción aleatoria.
- Con probabilidad $1 - \epsilon$ se escoge la mejor acción.

Por ejemplo, si tenemos dos acciones (A y B), siendo A la mejor, con $\epsilon = 0.5$:

- Con probabilidad 0.5 se escoge A.
- Con probabilidad 0.5 se escoge aleatoriamente entre B y A.

Comparativa entre métodos directos e indirectos

Métodos directos

La **política óptima** se encuentra **entrenando** la política **directamente**.

Métodos indirectos

Encontrar una **función de valor óptima** lleva a tener una **política óptima**.

Por lo tanto Independientemente del método, tendremos una política.

- Pero en el caso de los métodos basados en valores no la entrenamos.
- Será una "simple" función que usará los valores dados por la función v_π o q_π.

Q-learning

Técnica en la que se aprende una función (tabla) acción-valor o función Q:

- Entrada: Estado y acción a realizar.
- Salida: **Recompensa esperada** de esa acción (y de todas las posteriores).

La función Q se actualiza de forma iterativa:

1. Antes de explorar el entorno, Q da el mismo valor fijo (arbitrario).
2. Según se explora, aproxima mejor el valor de la acción a en un estado s.
3. Según se avanza, la función Q se actualiza.

Representa suma de las recompensas de elegir la acción Q y todas las acciones óptimas posteriores.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Realizar a_t en el estado s_t actualiza su valor con un término que contiene:

- α: Lo "agresivo" que sstamos haciendo el entrenamiento.
- r_t: Estimación que obtuvimos al actuar en el estado e_t anteriormente.
- $\max_a Q(s_{t+1}, a)$: Recompensa futura estimada (la que vamos aprendiendo).
- $\gamma \in [0, 1]$: El factor de ajuste que sube o baja la recompensa futura.
- Se resta además el valor antiguo para incrementar o disminuir la diferencia en la estimación.

Ahora tenemos una estimación de valor para cada par estado-acción.

- Con ella, podemos elegir la acción que nos interesa (e.g. usando ϵ-greedy)

Otras soluciones

Deep Q-networks (DQN)

Son aproximaciones de funciones Q utilizando redes neuronales profundas[2].

Asynchronous Advantage Actor-Critic (A3C)

Es una combinación de las dos técnicas anteriores[3], combinando:

- Un actor: Red de políticas de actuación que deciden qué acción tomar.
- Un crítico: DQN que decide el valor de cada acción a tomar.

2 https://www.nature.com/articles/nature14236
3 https://proceedings.mlr.press/v48/mniha16.html


0:00 / 1:02

Relevancia del aprendizaje por refuerzo

"El Go es un juego estudiado por los humanos durante más de 2500 años. AlphaZero, en un tiempo insignificante (3 días), pasó de conocer sólo las reglas del juego a vencer a los mejores jugadores del mundo, superando todo nuestro conocimiento acumulado durante milenios. Ningún campo del aprendizaje automático ha permitido avanzar tanto en este tipo de problemas como el aprendizaje por refuerzo."

Relevancia del aprendizaje por refuerzo hoy en día

Podemos decir que es prácticamente el único paradigma de aprendizaje:

- Capaz de aprender comportamientos complejos en entornos complejos.
- Que ha podido hacerlo prácticamente sin supervisión humana.

Ofrece a la robótica forma abordar cómo diseñar comportamientos difíciles.

- Que por otro lado, son prácticamente todos.
- Las cosas fáciles para un humano suelen ser las más complejas de diseñar.

Permite a robots descubrir de forma autónoma comportamientos óptimos:

- No se detalla la solución al problema, sino que se interacciona con el entorno.
- La retroalimentación de el efecto sobre el entorno permite aprender.

La utilidad de los modelos aproximados

Los datos del mundo real pueden usarse para aprender modelos aproximados.

- Mejor, porque el proceso de aprendizaje por ensayo y error es muy lento.
- Sobre todo en un sistema que tiene que hacerlo en un entorno físico.
- Las simulaciones suelen ser mucho más rápidas que el tiempo real.
- Y también mucho más seguras para el robot y el entorno
- ***Mental rehearsal***: Describe el proceso de aprendizaje en simulación.

Suele ocurrir que un modelo aprende en simulación pero falla en la realidad:

- Esto se conoce como **sesgo de simulación**.
- Es análogo al sobreajuste en el aprendizaje supervisado.
- Se ha demostrado que puede abordarse introduciendo modelos estocásticos.

Impacto del conocimiento o información previa

El conocimiento previo puede ayudar a guiar el proceso de aprendizaje:

- Este enfoque reduce significativamente el espacio de búsqueda.
- Esto produce una **aceleración** dramática **en el proceso de aprendizaje**.
- También **reduce la posibilidad de encontrar mejores óptimos**[1].

Existen dos técnicas principales para introducir conocimiento previo:

- A través de la **demostración**: Se da una política inicial semi-exitosa.
- A través de la **estructuración de la tarea**: Se da la tarea dividida.

[1] Alpha Go fue entrenado con un conocimiento previo de Go, pero Alpha Go Zero no sabía nada del juego. El resultado fue que Alpha Go Zero jugó y ganó a Alpha Go en 100 partidas.

Desafíos del aprendizaje por refuerzo

La maldición de la dimensionalidad: El espacio de búsqueda crece exponencialmente con el número de estados.

La maldición del mundo real: El mundo real es muy complejo y no se puede simular.

- Desgaste, estocasticidad, cambios de dinámica, intensidad de la luz, ...

La maldición de la incertidumbre del modelo: El modelo no es perfecto y no se puede simular.

- Cada pequeño error se acumula, haciendo que conseguir un modelo suficientemente preciso del robot y su entorno sea un reto

Algunas reflexiones

Es probable que una IA más avanzada requieran recompensas más complejas.

- Por ejemplo, un vehículo autónomo al principio puede estar ligada a algo tan simple como "llegar del punto a al punto b a salvo", pero...
  - ¿Y si se ve obligado a elegir entre mantener el rumbo y atropellar a cinco peatones o desviarse y atropellar a uno?
  - ¿Debe desviarse o incluso dañar al conductor con una maniobra peligrosa?
  - ¿Y si el único peatón es un niño, o un anciano? ¿una mujer? ¿un hombre? ¿un transexual? ¿la próxima Marie Curie? ¿el próximo Hitler? ¿un cuadro valiosísimo e irremplazable? ¿cambia eso la decisión? ¿por qué?

De repente el problema es mucho más complejo al intentar matizar la función objetivo

Dentro de la ética moral, una de las principales preguntas es: **¿qué debemos hacer?**

- ¿Cómo debemos vivir? ¿Qué acciones son correctas o incorrectas?

Nosotros los humanos, ¿tenemos funciones de valor? ¿qué nos motiva?

- Porque ojo, hay conceptos más complicados que el placer y el dolor como el bien y el mal, el amor, la espiritualidad, ...
- ¿Se podría al menos esbozar la recompensa que maximizamos en nuestra vida real?

Y como humanos, ¿cómo sabemos lo que es correcto o no? ¿Por intuición?

- Generalmente podemos responder que estos valores nos vienen "por intuición".
- Seguramente, pero poner la en palabras o reglas es sencillamente imposible.
- Y **probablemente una máquina pueda aprender estos valores de alguna manera**.
- Probablemente esto es uno de los problemas más importantes que os tocará resolver.

¡GRACIAS!