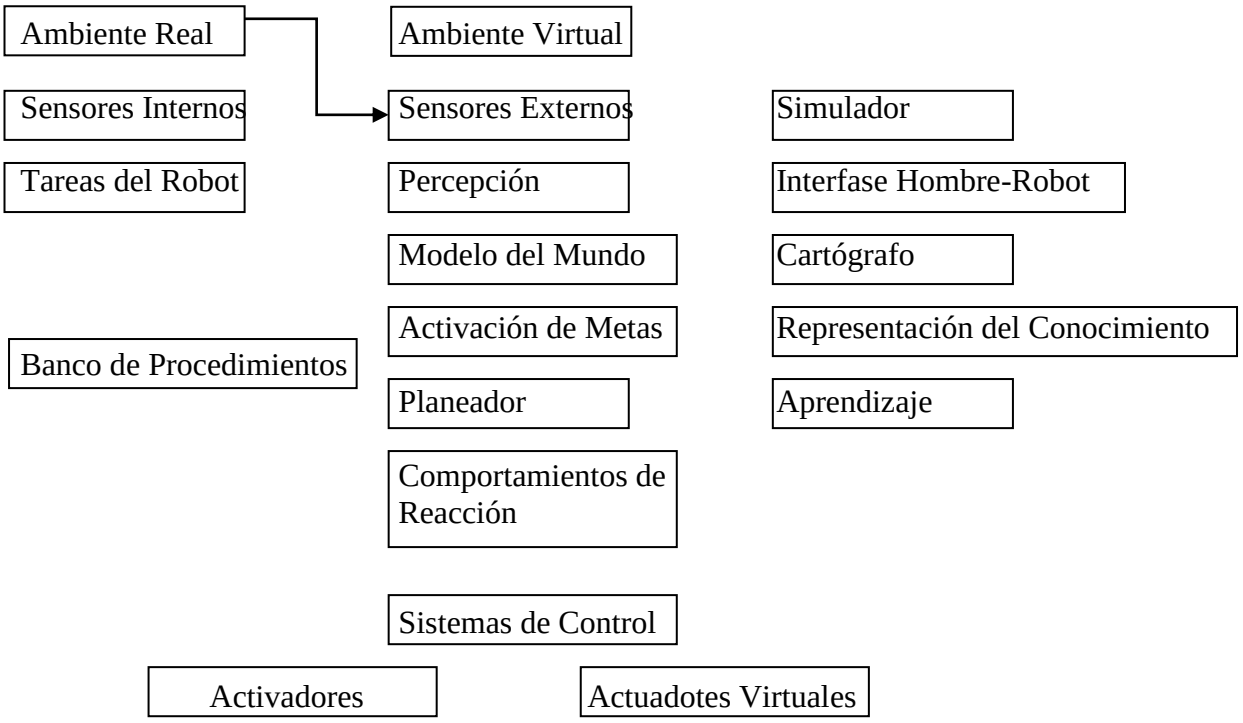
Robots Móviles y Agentes Inteligentes

Jesús Savage
savage@servidor.unam.mx

Máquina de Estado
- Entradas
- Máquina
- Salidas

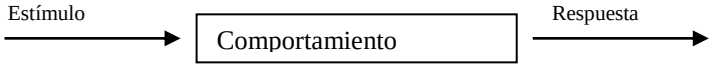
Máquina de Estado Aumentada
Una entrada (o salida) de una máquina puede ser sustituida por la salida de otra máquina de estados. Esa bloquea la entrada (o salida) ordinaria y pone su salida.

Examen Final
Proyecto
Tareas



Arquitecturas Reactivas (o por Comportamientos)

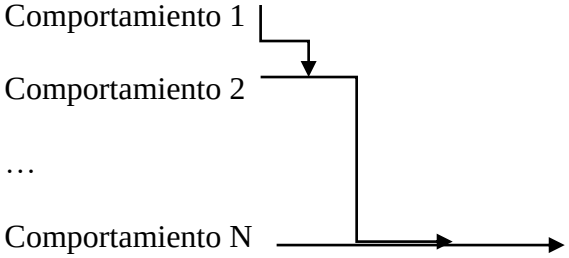
1. Diagramas de Stimulus Response (SR)



La idea es que la respuesta sea instantánea.
En los comportamientos puros, la respuesta únicamente depende del estímulo.

Organización de los comportamientos:

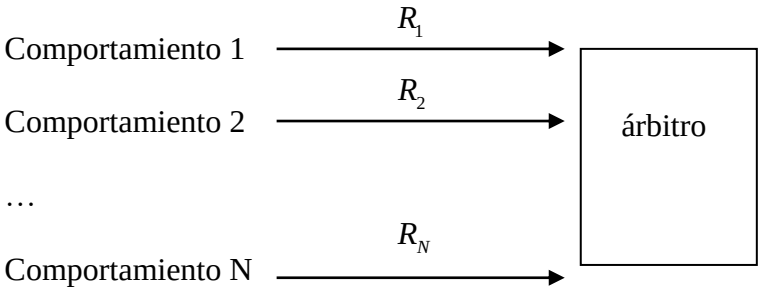
Esquema 1



Algunos comportamientos bloquean la salida de otros. Solamente la salida de uno es el que se usa, todos se activan. El diagrama anterior decide la prioridad.

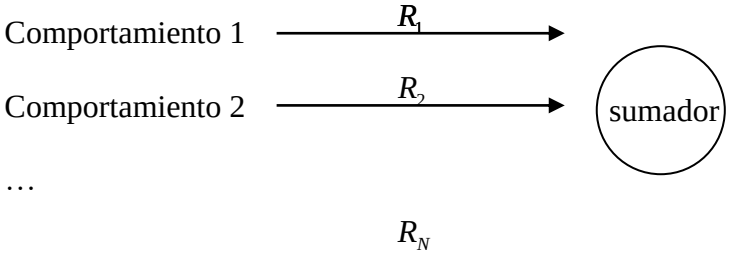
La salida generalmente es un vector que, por ejemplo, indica la magnitud y dirección del movimiento del robot.

Esquema 2



El árbitro decide cuál es la respuesta.

Esquema 3



Comportamiento N ⟶

El sumador hace un promedio ponderado por *ganancias*.

$$\sum_{i=1}^{N} g_i R_i$$

La ganancia depende del diseñador y va a hacer que el robot se comporte de diferente forma (robot "audaz" contra robot "conservador").

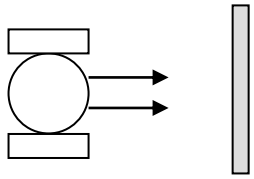
Un diagrama del espacio se puede hacer con campos vectoriales (diagramas de flechitas tipo sistemas dinámicos), por ejemplo uno que sea de las fuerzas de atracción y otro de repulsión y sumarlos.

Puedo tener una complejidad grande combinando comportamientos, pues pueden controlar diferentes cosas.

Máquinas de Estado
Algoritmo para el Comportamiento de Evadir Obstáculos

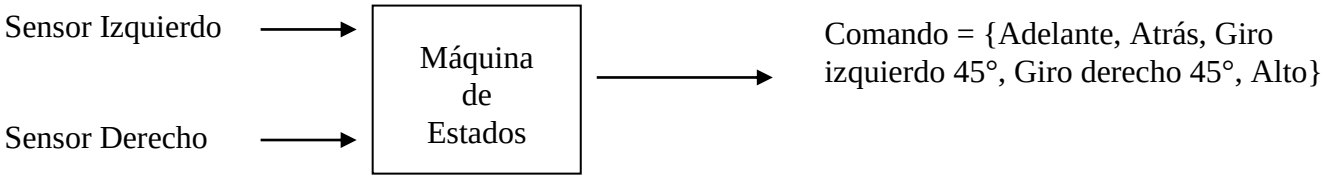
Un robot circular con:

- Dos motores, uno a cada lado
- Dos sensores adelante, para detectar obstáculos (por ejemplo, infrarrojos, ultrasonido, etc.)



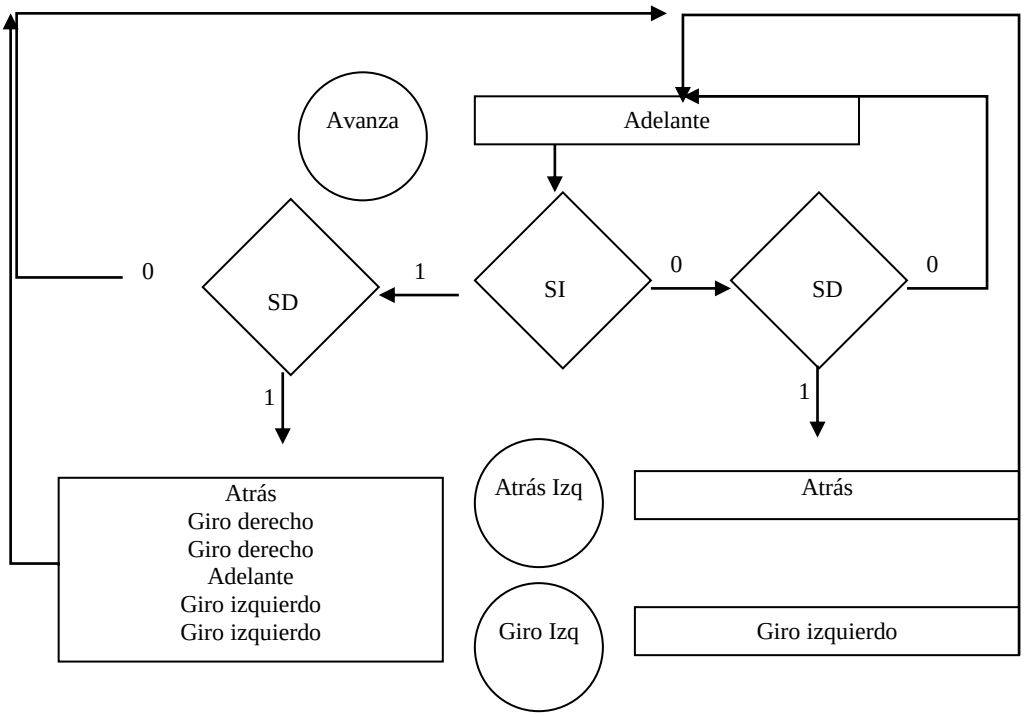
Obviamente:

- Si los sensores no detectan el obstáculo, seguir avanzando
- Si el sensor derecho lo detecta y el otro no, hacerlo tantito para atrás y girar hacia la izquierda para seguir avanzando
- Si el sensor izquierdo lo detecta y el otro no, hacerlo tantito para atrás y girar hacia la derecha para seguir avanzando
- Si los dos detectan, hacerlo para atrás y giro 90°.



CONVENCIÓN: Giros izquierdos = ángulos positivos

Notación de diagrama de flujo



Si se quisiera programar esto con una subrutina, hace todo. En la teoría de comportamientos necesito una respuesta inmediata y necesito que cuando vuelva a entrar se guarde el estado.

Simulador de Robots
ftp://rha.fi-b.unam.mx
anonymous:biyectivo@hotmail.com

cd pub/lii/Roc2

Tarea 0
Bajar el simulador y probarlo. Compilar el código del Roc2user.cpp
Tarea Pi: Hacer un prerrequisites document (máquinas de estado, conceptos de programación, apuntadores, etc.)

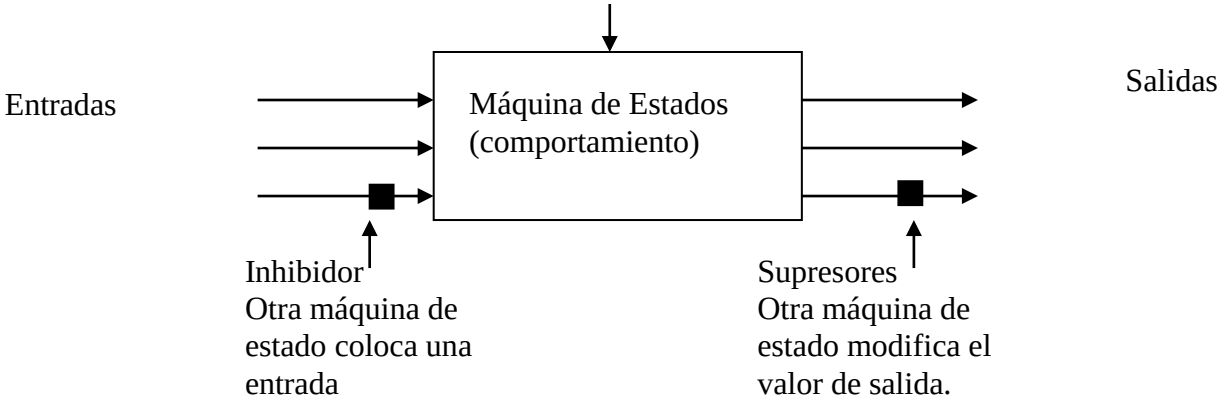
AFSM (Augmented Finite State Machine)

La máquina de estados aplicada a robots representa el comportamiento.
Una máquina tradicional de estados tiene entradas y salidas.

Rodney Brooks (MIT) inventó la máquina de estados finitos aumentada.

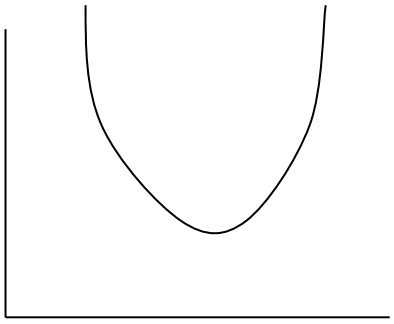
Básicamente el concepto es que alguna de las entradas es inhibida por otra máquina de estados (su salida se convierte la entrada) y en las salidas están los supresores, que bloquean la salida de la máquina y colocan un valor. Por último hay una entrada de "reset" que hace que se coloque en un estado en específico.

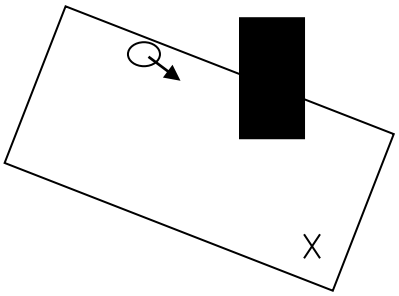
Con AFSM podemos tener diferentes capas y hacer una respuesta **jerárquica**.



Campos Potenciales

Se modela el robot como una canica. Los obstáculos ahora son montañas que ejercen fuerzas de repulsión. El destino es un hoyo.
El robot se mueve a través de un campo potencial por la pendiente más pronunciada hasta que lo lleva al destino.





$y = y_0 + (x - x_0)^2$

El mínimo se encuentra diferenciando.

$$\frac{\partial y}{\partial x} = 2(x - x_0)$$

$$x^* = x_0$$

Supongamos que no se conoce la ecuación; entonces se utiliza un método iterativo.
Empiezo con un punto x_{n-1} cualquiera y dependiendo del valor de la pendiente me muevo hacia la izquierda o la derecha.

Es una relación de recurrencia:

$x_n = f\left(x_{n-1}\right) = x_{n-1} - \delta \frac{\partial y}{\partial x}$ donde δ es una constante.

Por ejemplo:
Si consideramos $\delta = \frac{1}{2}$ se tiene $x_n = x_{n-1} - \frac{1}{2}\left(2\left(x_{n-1} - x_0\right)\right) = x_0$.

Esto da pie a considerar la situación general.

Se utiliza Steepest Descent.

Posición del robot en n: $q_n = \left[x_n, y_n\right]^T$

$q_n = q_{n-1} - \delta f\left(q_{n-1}\right)$

donde $f\left(z\right) = \dfrac{F\left(z\right)}{\left\|F\left(z\right)\right\|} = \nabla U\left(z\right)$

y $U\left(z\right)$ es el campo potencial del medio ambiente en donde navega el robot

$U\left(z\right) = U_{atr}\left(z\right) + U_{rep}\left(z\right)$ (dos componentes, uno atractor y un repulsor).

Al final se tienen fuerzas de atracción más fuerzas de repulsión.

Se define:

$U_{atr} = \frac{1}{2}\varepsilon_1 \left\|q - q_{dest}\right\|^2$ (campo de tipo parabólico) donde q_{dest} es la posición del destino.
Se ve claro que es parabólico:

$$U_{atr} = \tfrac{1}{2}\varepsilon_1\left[\left(x - x_{dest}\right)^2 + \left(y - y_{dest}\right)^2\right]$$

La fuerza de atracción en *q* es:

$$\nabla U_{atr} = \varepsilon_1\begin{bmatrix}\left(x - x_{dest}\right)\\ \left(y - y_{dest}\right)\end{bmatrix} = \varepsilon_1\left[q - q_{dest}\right] = F_{atr}\left(q\right)$$

La fuerza de atracción se va a convertir en aceleración. Para fuerzas muy grandes se transforma en aceleraciones muy grandes.
Esto no se quiere, por lo que se tiene un umbral. Esta fórmula solamente aplica para $\left\|q - q_{dest}\right\| \le d_1$.

¿Qué pasa en $\left\|q - q_{dest}\right\| > d_1$?

$$U_{atr} = E_2\left\|q - q_{dest}\right\|$$

Esto es un campo cónico:

$$U_{atr} = E_2\sqrt{\left(x - x_{dest}\right)^2 + \left(y - y_{dest}\right)^2}$$

$$\nabla U_{atr} = E_2 \begin{bmatrix} \dfrac{(x - x_{dest})}{\sqrt{(x - x_{dest})^2 + (y - y_{dest})^2}} \\ \dfrac{(y - y_{dest})}{\sqrt{(x - x_{dest})^2 + (y - y_{dest})^2}} \end{bmatrix} = \frac{E_2}{\|q - q_{dest}\|} \begin{bmatrix} x - x_{dest} \\ y - y_{dest} \end{bmatrix} = E_2 \frac{q - q_{dest}}{\|q - q_{dest}\|}$$

Campos Repulsivos
¿Cómo representar el obstáculo? Se podría considerar un solo vector en el centroide, una serie de puntos que ejercen repulsión, etc. Depende del diseñador.

$$U_{rep}(q) = \tfrac{1}{2}\eta \left(\frac{1}{\|q - q_{obs}\|} - \frac{1}{d_0} \right)^2$$

Esto aplica cuando $\|q - q_{obs}\| \le d_0$. En caso contrario, se toma como 0. Esto es para que no lo tomes en cuenta si está demasiado lejos.

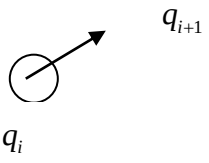
$$\nabla U_{rep}(q) = -\frac{\eta}{\|q - q_{obs}\|^2} \left(\frac{1}{\|q - q_{obs}\|} - \frac{1}{d_0} \right) \left(\frac{q - q_{obs}}{\|q - q_{obs}\|} \right) = F_{rep}(q)$$

En general,

$$F(q) = F_{atr}(q) + \sum_{i=1}^{n} F_{rep}(q)$$

$$f(q) = \frac{F(q)}{\|F(q)\|} = \nabla U(q)$$

$$q_{i+1} = q_i - \delta_i f(q_i)$$



EJEMPLO

Robot: (1,1)
Obstáculo: hexágono centrado en (2,2)
Objetivo: (5,4)

$q_0 = (1,1)$
Se va a tomar $d_0 = 5, \varepsilon_1 = 1, \eta = 2, \delta_0 = 1$ y se va a utilizar la parabólica.

Se tiene:

$$F_{atr}(q_0) = \varepsilon_1 (q_0 - q_{dest}) = (-4,-3)$$

$$F_{rep}(q_0) = -\frac{\eta}{\|q_0 - q_{obs}\|^2}\left(\frac{1}{\|q_0 - q_{obs}\|} - \frac{1}{d_0}\right)\left(\frac{q_0 - q_{obs}}{\|q_0 - q_{obs}\|}\right)$$

$$= \frac{-2}{\sqrt{2}}\left(\frac{1}{\sqrt{2}} - \frac{1}{5}\right)\left(\frac{1}{2\sqrt{2}}\right)(-1,-1)$$

$$= (.3585,.3585)$$

La fuerza total es:

$$F(q_0) = (-3.64,-2.64) \qquad f(q_0) = (-0.8091,-0.5868)$$

Finalmente:

$$q_1 = q_0 - \delta_0 \; f(q_0)$$
$$= (1,1) + (0.8091, 0.5868)$$
$$= (1.8091, 1.5868)$$

Obsérvese que no hemos considerado la **orientación inicial del robot**.

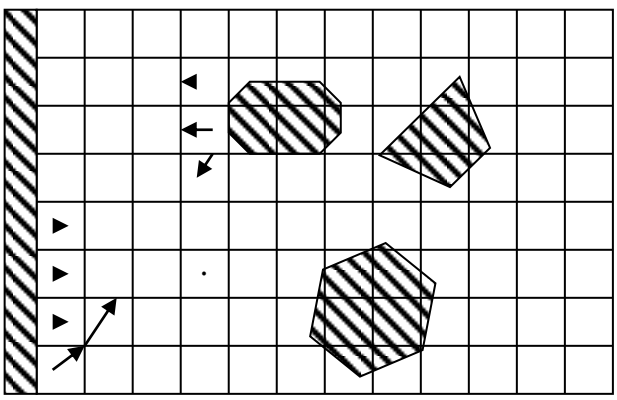
¿Cómo encontrar las constantes? Empíricamente

Nota: Si no conocemos obstáculos no podemos calcular las fuerzas de repulsión. En estos casos, se tiene que usar la información de los sensores para identificar al obstáculo y, con base en eso, calcular estas fuerzas.

Cálculo de los vectores de repulsión usando la posición de los obstáculos fijos

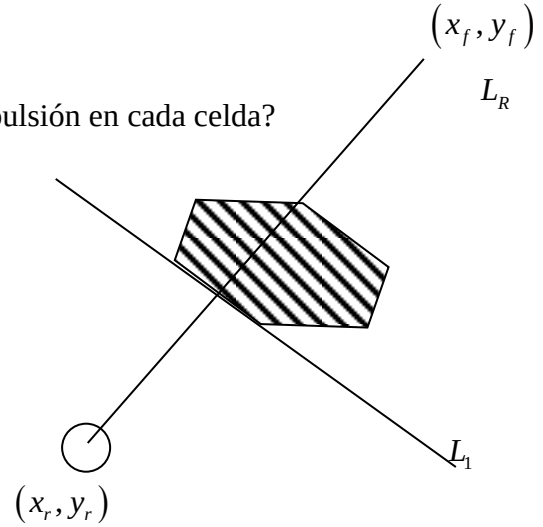
Los obstáculos conocidos se representarán por polígonos que tienen nodos ordenados en el sentido de las manecillas del reloj.

El espacio se divide en celdas y en cada celda se calcula la fuerza total de repulsión.



Etc.

¿Cómo encontrar la fuerza de repulsión en cada celda?



El punto (x_r, y_r) es el centro de la celda en donde se está (se asume que el robot está en el centro de la celda). El punto (x_f, y_f) es un punto arbitrario (se quiere calcular para la mayor cantidad de direcciones posibles. No es en línea sino que se hace offline).

$L_1 : y = m_1 x + b_1$
$L_R : y = m_R x + b_R$

Nótese que
$x_f = d\cos\phi + x_r$
$y_f = d\sin\phi + y_r$
Donde el ángulo es entre el eje *x* y la línea L_R.

Obs: se puede encontrar

$$m = \frac{y_1 - y_0}{x_1 - x_0} \qquad b = \frac{x_1 y_0 - y_1 x_0}{x_1 - x_0}$$

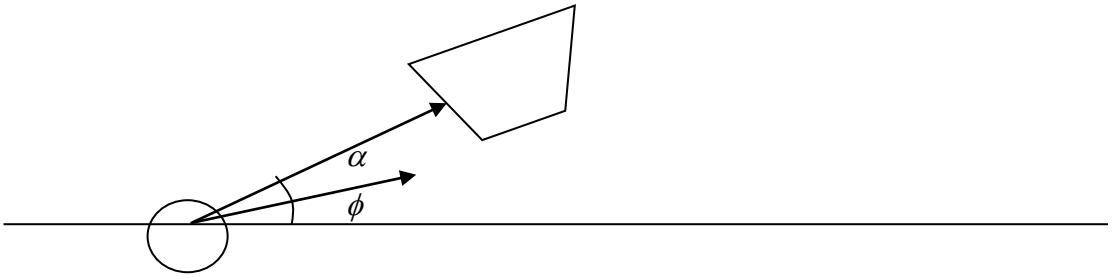
Los puntos de intersección son

$$x_{\text{int}} = \frac{b_r - b_1}{m_1 - m_r} \qquad y_{\text{int}} = m_r x_{\text{int}} + b_r$$

Obsérvese que se tiene que cumplir que $x_{\text{int}} \in (x_0, x_1)$ y $y_{\text{int}} \in (y_0, y_1)$.

Cómo calcular el campo potencial para el caso de objetos desconocidos

Lo anterior se hace offline y se alimenta al robot. Si hay objetos desconocidos:



El ángulo theta es el eje principal del robot. El ángulo alpha es el ángulo entre el eje y el sensor.

$$x_{obs} = d\cos(\phi + \alpha)$$
$$y_{obs} = d\sin(\phi + \alpha)$$

$$q - q_{obs} = (0,0) - (x_{obs}, y_{obs}) = (-x_{obs}, -y_{obs})$$

(puesto que se cambia el eje de coordenadas, verificar).

Tarea 1
Usar Roc2. Poner obstáculos *pickable* en el medio. Programar un robot para encontrar esos obstáculos y que los lleve a un lugar. Usar nada más la parte rectangular del mundo. El robot se mueve aleatoriamente. Tener una pila que se vaya bajando.

DATOS DE ROC2

Datos sobre esto:

- El Roc2 tiene comandos de scripting en el directorio de /scripts. Ver pick.txt. En un script se puede poner los objetos para ya no tener que teclearlos.
En el archivo también: *@xxx* es el identificador del comando.
Para ejecutar: **exe pick.txt**.

Para poner un objeto pickable
Reloj pick nombre x y colorR colorG colorB @xxx

El mundo está en lii.wrl.

En el script también se pueden llamar funciones.

Para tomar objetos:

grab – toma objetos
release – deja objetos

Para cargar un mundo, load <nombredemundo.wrl>
Los mundos están en maps

- Los vértices del polígono que representa el objeto son x_0, y_0 hasta x_3, y_3.

El archivo del mundo:

```
limit_area living-room 0.00 0.00 0.00 77.00 74.50 77.00 74.50 0.00
```

(son las coordenadas de las esquinas)

```
dimensions living-room 74.50 77.00
```

Es la coordenada más grande.

```
scale_expansion 1.0 1.2
wall_expansion 1.5
```

Es el ancho que se tiene que añadir a cualquier cálculo para considerar el ancho del robot a la hora de pasar cerca de las paredes y objetos.

```
Middle_point living-room 15. 30.
```

Punto medio del cuarto

```
Polygon object living-room caja1 20. 30. 20. 45. 25. 45. 25. 30.
```

Posición del objeto

```
Polygon wall living-room pared1 0.00 0.00 0.00 1.50 74.50 1.50 74.50 0.00
```

Posición de una pared

```
Door living-room 65.40 1.50 65.40 3.20 74.50 3.20 74.50 1.50
```

Una puerta

Polygon pickable obj1 <coordenadas>

Dada la posición donde se encuentra el robot (x_R, y_R, θ) queremos saber la distancia del robot al objeto.

Dado el centroide y el tamaño se pueden encontrar las coordenadas de los vértices.

La función para el sensor:
shs(sonar, número_de_sensor)

NO EXISTE UN SENSOR QUE DETECTE A LOS PICKABLES EN ESTA VERSIÓN.
Hacer una función myshs_pickable() para detectar los pickable.

Se debe hacer con campos potenciales para evadir objetos inesperados para no chocar. En caso que choque, debe evadir con máquinas de estado. (Los obstáculos fijos sí los detecta shs(sonar, número_de_sensor).

Qué hacer, en resumen:
1. Hacer una función de usuario con dos comportamientos
1.1 Hacer una máquina de estados para evadir obstáculos fijos (al principio evadir los pickables, después evadir cualquier obstáculo). Ya hay una que es *user4* pero se podría modificar.

(posteriormente) 1.2 Comportamiento con campos potenciales. Hacer esta en *user7* (o cualquier número mayor).

Las ecuaciones habrá que programarlas (incluyendo: dado un obstáculo encontrar la distancia y con ello encontrar la posición *xy* del obstáculo, usando los sensores).
Un pickable se convertirá en atractor y si se encuentran otros objetos son los repulsares.

2. Función de sensores para detectar los pickables
La información está dada en decímetros.
Usar un umbral de 5 decímetros. Si la distancia entre el robot y el pickable es menor a 5 dm, me da la distancia; de otra forma me da 0.

Si está mayor a una cierta distancia, regresar que no lo detecta.

3. Comportamiento de manera aleatoria. Navegar de manera aleatoria hasta detectar un pickable. Llegar a la zona evadiendo los otros pickables. Esa zona se debe detectar con otra zona.

Usar todos los sensores para buscar, pero una vez que ya lo encontré con el sensor, giro para que el sensor de enfrente se alinee.

El sonar NO DEBE TOMAR el centroide para ver si está cerca. Tiene que encontrar la intersección con la línea.

En las subrutinas el resultado debe ser instantáneo y se debe guardar el estado en el que está. La precedencia no programarla con árbitros sino dejar que sea el orden de programación.

¿Qué se quiere?
De $(x_{i-1}, y_{i-1}, \phi_{i-1})$ (donde está el robot) necesito llegar a (x_i, y_i, ϕ_i).
Obsérvese que

$$x_i = x_{i-1} + x_i^{'} \cos(\phi_i)$$
$$y_i = y_{i-1} + x_i^{'} \sin(\phi_i)$$
donde

$$\phi_i = \phi_{i-1} + \phi_i^{'}$$
$$\phi_i = \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right)$$

Esto se conoce como **computación directa**.

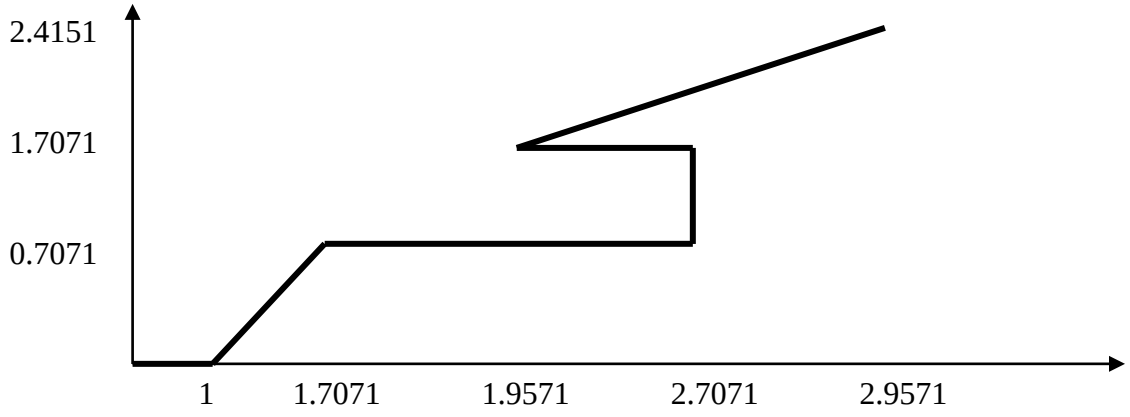
Otra opción es:

$$x_i^{'} = \frac{(x_i - x_{i-1}) + (y_i - y_{i-1})}{\cos(\phi_i) + \sin(\phi_i)}$$
$$\phi_i^{'} = \phi_i - \phi_{i-1}$$

y obviamente también se tiene que $x_i^{'} = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$

Esto se llama **computación inversa** y es la que nos interesa.

Ejemplo



Sup: Los puntos los encontré con campos potenciales (recordar que el campo potencial da el vector de movimiento:

$$q_i = (x_i, y_i) = q_{i-1} - \delta f(q_{i-1})$$

i=1

$x_0 = 0 \;\; y_0 = 0 \;\; \phi_0 = 0$

$x_i = 1 \;\; y_i = 0$

Por lo tanto: $\phi_1 = \arctan\left(\frac{y_1 - y_0}{x_1 - x_0}\right) = 0$ y $x_1^{'} = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} = 1$. Y finalmente: $\phi_i^{'} = \phi_i - \phi_{i-1} = 0$

Ad nauseam.
Los valores son:

i	$\phi_i^{'}$	$x_i^{'}$
1	0°	1
2	45°	1
3	-45°	1
4	90°	1
⋮	⋮	⋮

Trayectorias

El robot casi siempre avanza en línea recta.
¿A qué velocidad debe ir el robot?

El robot está en $(x_{i-1}, y_{i-1}, \phi_{i-1})$ y quiero llegar a (x_i, y_i, ϕ_i). ¿Cómo voy a recorrer la distancia $x_i^{'}$?
Tengo t_0 y el tiempo t_f (final).

Se quiere una función $f(t)$ que sea suave. Se asume que el robot está parado en ambos extremos.

$f(0)=0,\ \ f(t_f)=x_i^{'}$ [posiciones inicial y final]

$f'(0)=f'(t_f)=0$ [velocidades inicial y final]

Asumiendo una relación cúbica:

$f(t)=a_0+a_1t+a_2t^2+a_3t^3$

$f'(t)=a_1+2a_2t+3a_3t^2$

$f''(t)=2a_2+6a_3t$

Se encuentran:

$a_0=0, a_1=0, a_2=\frac{3x_i^{'}}{t_f^2}, a_3=-\frac{2x_i^{'}}{t_f^3}$. La ecuación es entonces:

$$f(t)=\frac{3x_i^{'}}{t_f^2}t^2-\frac{2x_i^{'}}{t_f^2}t^3$$

$$f'(t)=\frac{6x_i^{'}}{t_f^2}t-\frac{6x_i^{'}}{t_f^2}t^2$$

$$f''(t)=\frac{6x_i^{'}}{t_f^2}-\frac{12x_i^{'}}{t_f^2}t$$

Ejemplo

Suponer $t_f = 3 \;\; x_1^{'} = 1$

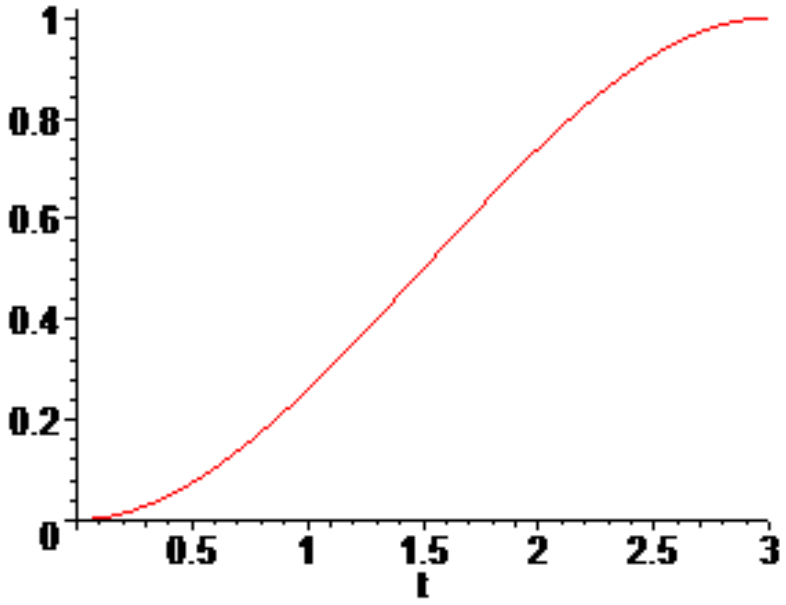
Entonces

$$f(t) = \frac{1}{3}t^2 - \frac{2}{27}t^3$$

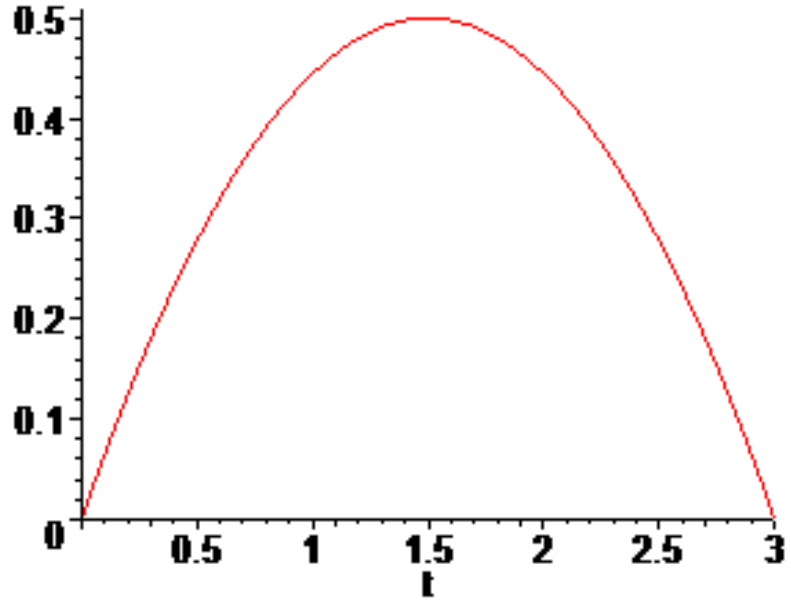
$$f'(t) = \frac{2}{3}t - \frac{6}{27}t^2$$

$$f''(t) = \frac{2}{3} - \frac{12}{27}t$$

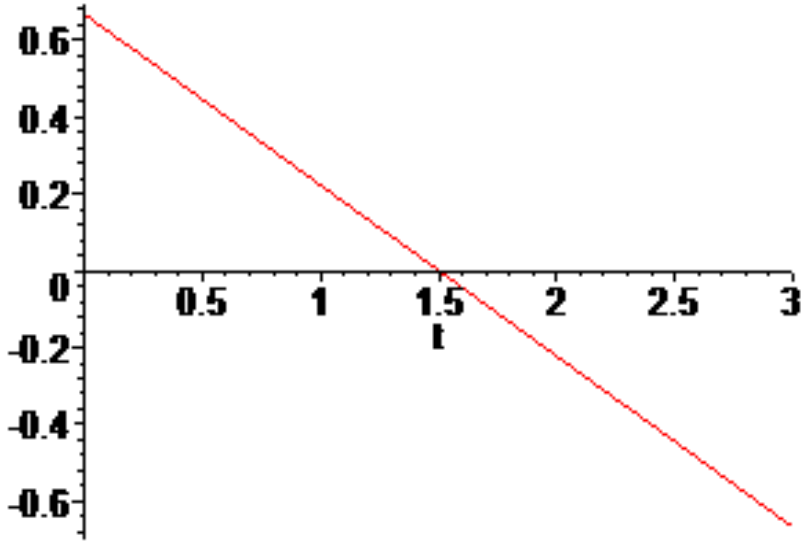
$f(t)$



$f'(t)$



$f''(t)$



En Roc2 para mover es:
mv x_i' ϕ_i' t_f
donde la primera está dada en decímetros y el ángulo en radianes y el tiempo en segundos.

En Roc2 las funciones cinemáticas se encuentran en el archivo Roc2UserMv.

Cómo relajar el supuesto de la velocidad final 0
Suavizar la trayectoria.

Una forma: Le ajustas una parábola a la esquinita
$y = y_0 + (x - x_0)^2$ con $x_b \leq x \leq x_f$ puntos pre-escogidos.

Otra forma: linearizar (segmentitos de recta).

Rotar y trasladar

polygon type-object x0 y0 x1 y1 x2 y2 … yn yn)
position *room* type-object name xc yc theta) ← esta función traslada a xc, yc y rota theta

Type-object define un objeto genérico, podemos definir varios objetos (de ese tipo).

Dados $\left(x_p, y_p\right)$

$$x = x_p \cos\alpha - y_p \sin\alpha + x_c$$
$$y = y_p \cos\alpha + x_p \sin\alpha + y_c$$

Donde x_c, y_c es el punto fijo sobre el que se está girando.

Ejemplo

Por ejemplo:

Polygon mesa1 0 0 0 2 2 2 2 0

Position salon mesa1 mesa_principal 4 4 $\frac{\pi}{4}$

Entonces:

i	x_i	y_i
0	4	4
1	2.59	5.414
2	4	6.28
3	5.4142	5.4142

Ancho del robot

Se necesita considerar el ancho del robot. Por lo menos, ensanchar los obstáculos conocidos en el radio del robot.
¿Cómo hacer con los desconocidos? De la lectura del sonar, restarle el radio del robot.

Escalamiento de polígonos

Suponer $P = \{(0,2),(1,3),(2,2),(1,1)\}$ un polígono.

1. Encontrar el centroide
2. Moverlo al origen
3. Multiplicar
4. Volverlo a mover al centroide.

Con $\lambda = 2$:

$$P_{esc} = \lambda\left(P - \mathbf{C}\right) + \mathbf{C}$$

$$P = \{(-1,2),(1,4),(3,2),(1,0)\}$$

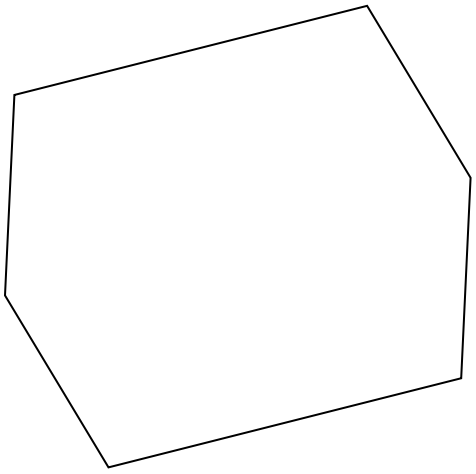
Siguiente tarea
El mundo tiene mesas y otros dos objetos tipo, puestas y rotadas.
Objetos fijos: sillones, mesas, cajas, etc. (al menos 3)

Descomponer el mundo en celdas.

Calcular los vectores de repulsión en cada celda.
Si el centroide de la celda queda dentro del obstáculo, está ocupada
Hacer una "estrella" hacia todas las direcciones y calcular si la intersección con algo es menor o igual a d0. La repulsión total es la suma.

Tres archivos: 1) objetos tipo 2) posiciones objetos réplica 3) archivo final .WRL con la descripción del mundo.

Verificación de si un punto está dentro o fuera de un polígono



Ecuación paramétrica de la recta

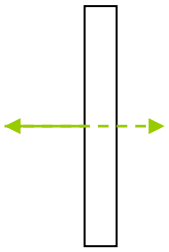
$$x = x_0 + (x_1 - x_0)t$$
$$y = y_0 + (y_1 - y_0)t$$
$$t \in (0,1)$$

Los puntos de la ecuación general

$$(y_0 - y_1)x + (x_1 - x_0)y + (x_0 y_1 - x_1 y_0) = 0$$

Puntos que cumplen esta ecuación, están en la línea.

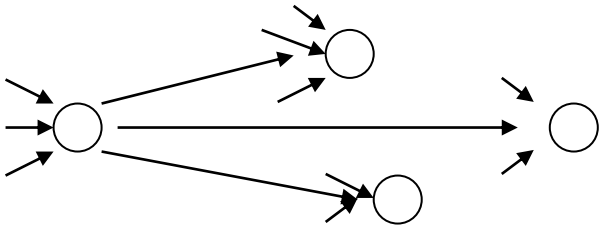
Puntos que evalúan >0, caen a la izquierda.
Puntos que evalúan <0, caen a la derecha.
Problema de los campos potenciales
Por ejemplo un mínimo local



El vector resultante es 0 y el robot se queda inmóvil o en un ciclo.

Comportamientos con Redes Neuronales

Redes neuronales: 1943 → McCullock
Una red neuronal es un modelo matemático de cómo funciona una neurona.
Una neurona está conectada a otras neuronas.



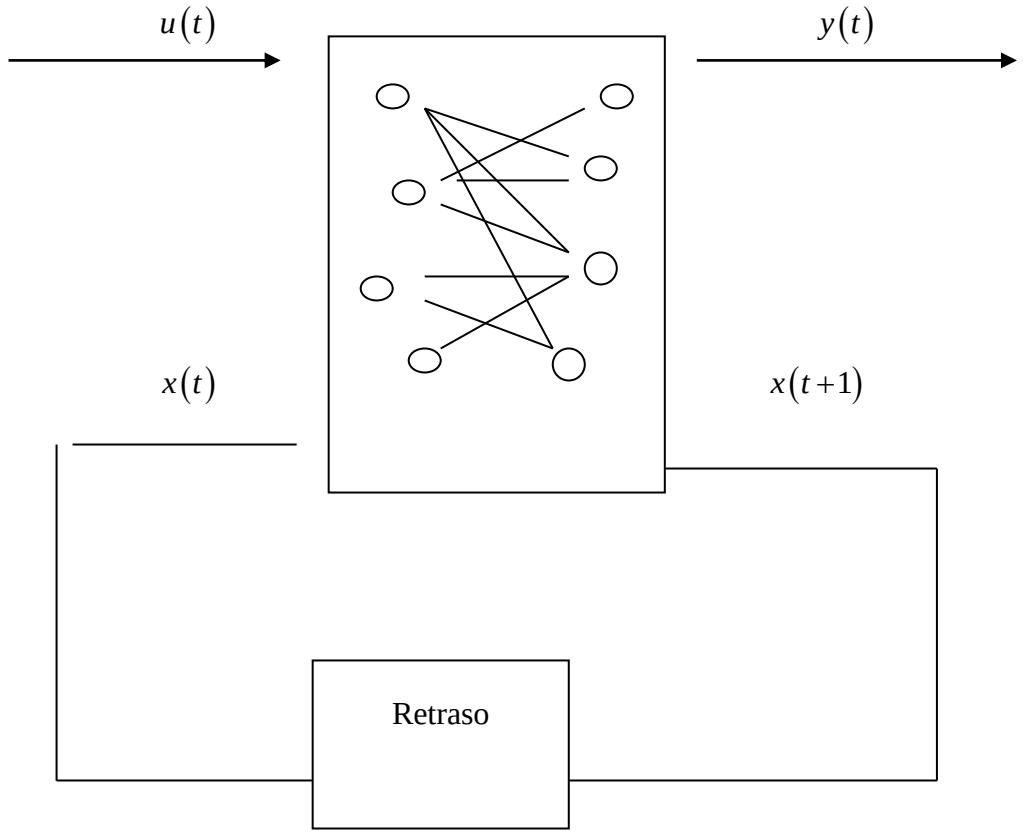
La entrada a la red neuronal es una imagen (por ejemplo, la cámara del robot). Se puede entregar la información en bruto o procesada.

Las conexiones se van uniendo por medio de aprendizaje.

Se entrena a la red para que siga los comportamientos, por ejemplo un coche autónomo. Para entrenarlo, se puede poner a un conductor y hacer que la red aprenda los comportamientos de acuerdo a las imágenes.
La respuesta de la red es: para una cierta imagen, se giró a la izquierda violentamente; para otra imagen, se giró ligeramente; etcétera.

También se puede hacer con información de sonares, etc.

Puedo hacer máquinas de estado con redes neuronales.



Modelo

Por cada neurona se tiene

$$u = \sum_{i=1}^{n} w_i y_i + \theta$$

Entradas: y_j → vienen de otras neuronas
Pesos: w_j
Umbral de activación: θ

La salida de la neurona es la *función de activación*

$$a = f(u)$$

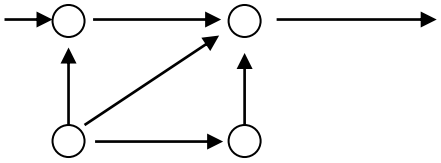
Generalmente $f(u) = \frac{1}{1+\mathbf{e}^{-\frac{u}{T}}}$ (un *sigmoide*)

T es una constante de temperatura.

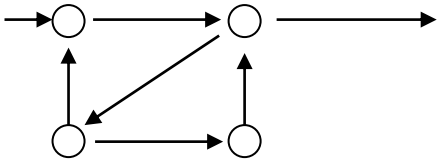
Resulta que f es la solución de la ecuación diferencial

$$\frac{dy}{du} = \frac{y(1-y)}{T}$$

Topología de Redes Neuronales
¿Qué pasa con la topología de redes neuronales (muchas neuronas)?



Topología acíclica (no hay retroalimentación de la salida hacia otra interna) → el dígrafo es acíclico



Topología cíclica (con retroalimentación) → el dígrafo es cíclico

Estos fueron los primeros modelos de redes neuronales.

Modelo del Perceptrón

$$u(x) = \sum_{i=1}^{n} w_i x_i + w_0$$

y $$y(x) = \begin{cases} 1, u(x) \geq 0 \\ 0, u(x) < 0 \end{cases}$$

Este modelo es utilizado para *detección y clasificación*.
Si las características de la entrada x es cercano a w tal que su producto interno $x^T w$ es mayor que un umbral $-w_0$, entonces la salida es 1, indicando la detección de un objetivo.

Ejemplo
Dos neuronas, una que representa una mesa y otra una silla.
Las dos tienen las mismas entradas, que son características invariantes a distancia y orientación. La salida debe de ser 1 para la de la silla y 0 para la de la mesa.

Entrenar a una red neuronal es encontrar los pesos.

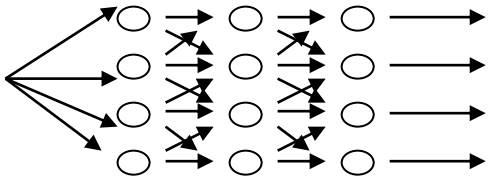
Minsky y Shannon

Shannon propone la matemática booleana y notación binaria en las computadoras.
Wiener inventó el término *cibernética* (el estudio de los sistemas – eléctricos, mecánicos, sociales, etc. – y su control).

LEER ASIMOV Y PHILIP DICK

Perceptrón multicapas

Capa de entradas
Capa intermedia
Capa de salida

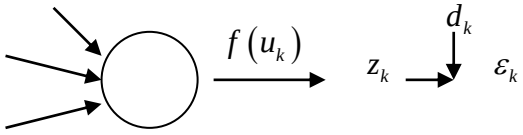


La capa de entrada tiene M neuronas.
La capa intermedia se conoce como *capa oculta* y tiene H neuronas.
La capa de salida tiene N neuronas.

El objetivo es encontrar los pesos que minimicen el error cuadrático medio.

Cada salida de las neuronas de salidas es un objetivo de clasificación.

Ejemplo
¿Cómo encontrar los pesos?
Tres entradas, una neurona de entrada.



$u = Wx$ donde $x = \begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}^T$ y $W = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}$

Encontrar unos pesos que dado un objetivo d_k se vaya actualizando con la característica que $\varepsilon \to 0$. Se tendrán *K* muestras de entrenamiento (x_k, d_k); salidas z_k y un error total $E = \sum_{i=1}^{K} \varepsilon_k^2$, donde $\varepsilon = d_k - z_k$.

$d_k \in (0,1)$ generalmente.

Obsérvese que $z_k = f(Wx_k)$.

El objetivo final es minimizar la función E respecto a W.

La solución es no lineal si *f* es una función sigmoide, por lo que generalmente se realiza la minimización con iteraciones:

$$W_{t+1} = W_t + \Delta W_t$$

Por método de *steepest descent*:

$$\Delta W_t = -\eta \frac{dE}{dW}$$

Solución

$$\frac{dE}{dw_i} = 2\sum_{k=1}^{K}\left(d_k - z_k\right)\left(-\frac{dz_k}{dw_i}\right)$$

Obsérvese que

$$-\frac{dz_k}{dw_i} = \frac{df\left(u\right)}{dw_i} = \frac{df\left(u\right)}{du}\frac{du}{dw_i} = f'\left(u\right)\frac{du}{dw_i} = f'\left(u\right)x_i$$

Por lo que

$$\frac{dE}{dw_i} = -2\sum_{k=1}^{K}\left(d_k - z_k\right)f'\left(u\right)x_i$$

Se denota $\delta_k = \left[d_k - z_k\right]f'\left(u_k\right)$ y entonces

$$\boxed{w_i\left(t+1\right) = w_i\left(t\right) + \eta\sum_{i=1}^{K}\delta_k x_i\left(k\right)}$$

En particular si $f\left(u\right)$ es el sigmoide, entonces

$$\delta_k = \left[d_k - z_k\right]z_k\left(1 - z_k\right)\alpha \;:=\; \frac{\partial E}{\partial u}$$

Prueba:

$\frac{\partial E}{\partial u} = -2\sum_{k=1}^{K}\left[d_k - z_k\right]\frac{dz_k}{du}$ y además $\frac{dz_k}{du} = \frac{f\left(u\right)\left[1 - f\left(u\right)\right]}{T}$. Por lo tanto,

$$\frac{\partial E}{\partial u} = -\frac{2}{T}\sum_{k=1}^{K}\left[d_k - z_k\right]f\left(u\right)\left[1 - f\left(u\right)\right] = -\frac{2}{T}\sum_{k=1}^{K}\left[d_k - z_k\right]z_k\left[1 - z_k\right].\ \text{QED}$$

El proceso se debe hacer hasta que $E < \varepsilon$.

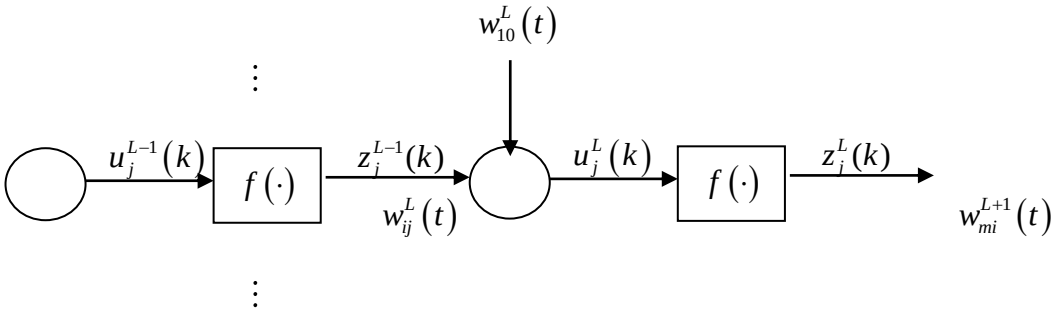
Cómo extender para un perceptrón multicapas

Se tienen:

K muestras de entrenamiento
L capas

$w_{ij}^{L}(t)$ es el peso de la neurona *i* a la neurona *j* en la capa L y en la iteración *t*.

Propagación de error:



El objetivo es (de nuevo) encontrar w_{ij} de cada capa que minimicen el error cuadrático medio.

$$\frac{dE}{dw_{ij}^{L}} = -2\sum_{k=1}^{K}\frac{dE}{w_{ij}^{L}} = -2\sum_{k=1}^{K}\frac{dE}{du_i^{L}(k)}\frac{du_i^{L}(k)}{dw_{ij}^{L}} = -2\sum_{k=1}^{K}\delta_i^{L}(k)\frac{du_i^{L}(k)}{dw_{ij}^{L}}$$

Pero dado que

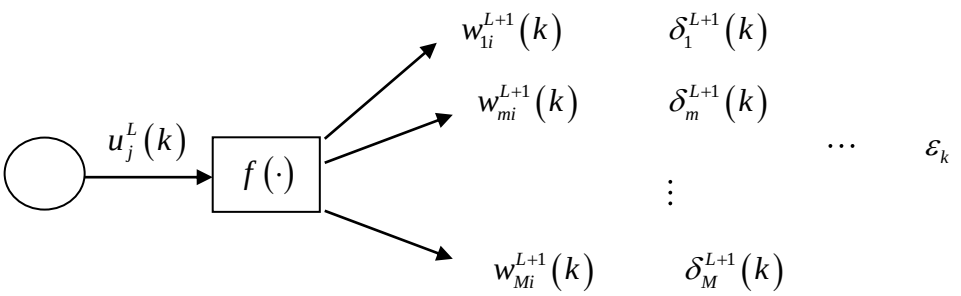
$$\frac{du_i^{L}(k)}{dw_{ij}^{L}} = \frac{d}{dw_{ij}^{L}}\sum_{m=1}^{M} w_{im}^{L} z_m^{L-1}(k)$$

Se puede reescribir la parcial como:

$$\frac{dE}{dw_{ij}^{L}} = -2\sum_{k=1}^{K}\delta_i^{L}(k)\, z_j^{L-1}(k)$$

Se observa que el error δ_i^{L} en esta capa depende del error en la capa anterior, por lo que se necesita utilizar una derivación tipo Bellman para tener:

$$\delta_i^{L}(k) = \frac{\partial E}{\partial u_i^{L}(k)}$$



Obsérvese que se tiene la siguiente recursión:

$$\begin{aligned}\delta_i^L(k) &= \frac{\partial E}{\partial u_i^L(k)} = \sum_{m=1}^{M} \frac{dE}{du_m^{L+1}(k)} \frac{du_m^{L+1}(k)}{\partial u_i^L(k)} \\ &= \sum_{m=1}^{M} \delta_m^{L+1}(k) \left[\frac{d}{\partial u_i^L(k)} \sum_{j=1}^{J} w_{mj}^L f\left(u_j^L(k)\right) \right] \\ &= f'\left(u_i^L(k)\right) \sum_{m=1}^{M} \delta_m^{L+1}(k) w_{mi}^L\end{aligned}$$

Esto se conoce como *backward propagation*.

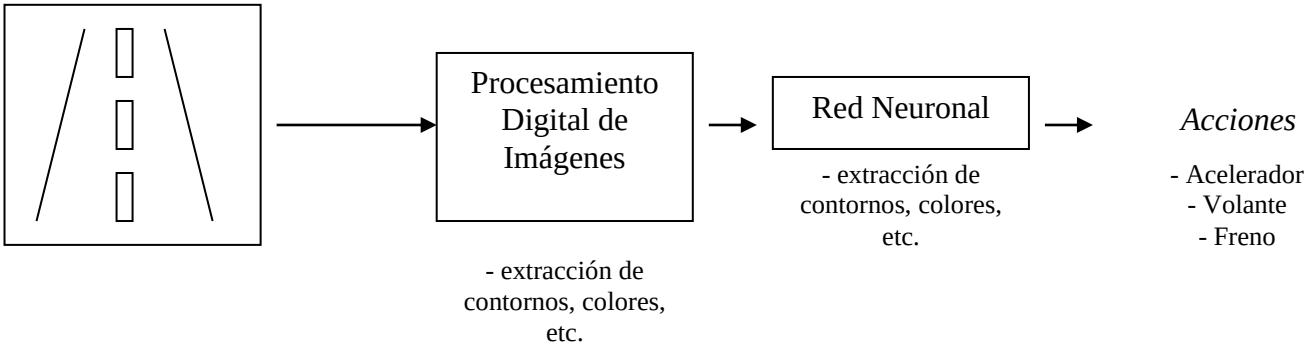
La fórmula de actualización será entonces:

$$w_{ij}^L(t+1) = w_{ij}^L(t) + \eta \sum_{k=1}^{K} \delta_i^L(k) z_j^{L-1}(k) + \underbrace{\mu\left[w_{ij}^L(t) - w_{ij}^L(t-1)\right]}_{\text{momento}} + \underbrace{\varepsilon_{ij}^L(t)}_{\text{ruido aleatorio}}$$

La η da la velocidad de convergencia. Sin embargo, si es muy grande, se puede dar la situación que se oscile demasiadas veces antes de llegar al mínimo.
El error aleatorio se suma para encontrar el mínimo global.

¿Cómo se usa esto para los robots?

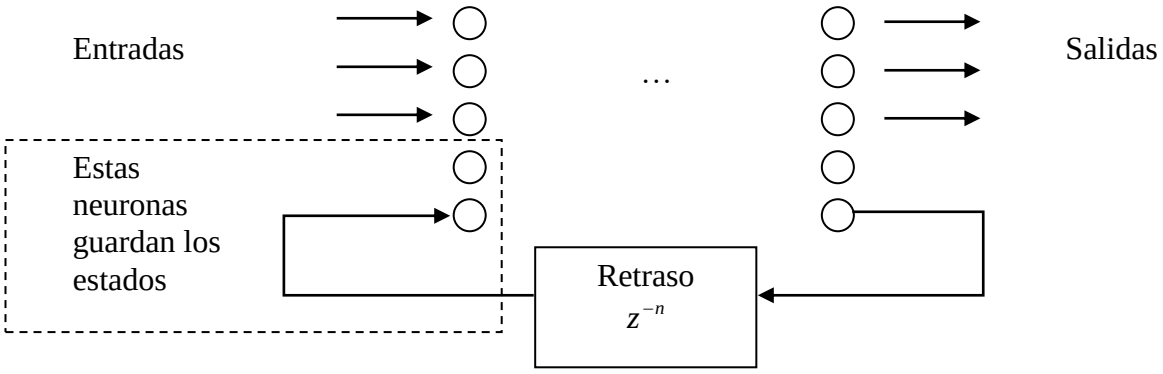
En Carnegie-Mellon University, se realizó un experimento en el cual un robot pudo manejar 90% autónomamente en una autopista larga de Estados Unidos.



El entrenamiento se haría primero manejando un humano y la red registrando qué se hizo y la foto correspondiente.

Otra forma: máquina de estado con redes neuronales.

Máquina de Estados con Redes Neuronales



Aprendizaje

El aprendizaje en un robot puede ser varias cosas:

1. Aprender "audacia": esto depende de las constantes (afinación de constantes)
Para campos potenciales: $\mu, \delta, d_1, d_0, \eta$

2. Aprender nuevos comportamientos
a) Crear nuevos algoritmos de máquinas de estado.
b) Modificación de las máquinas de estado

Hay dos cosas: una, lo que aprendemos, y otro, el conocimiento que se obtiene a través de la evolución. Por ejemplo, un bebé sabe cómo evitar un obstáculo, no tiene que aprender.

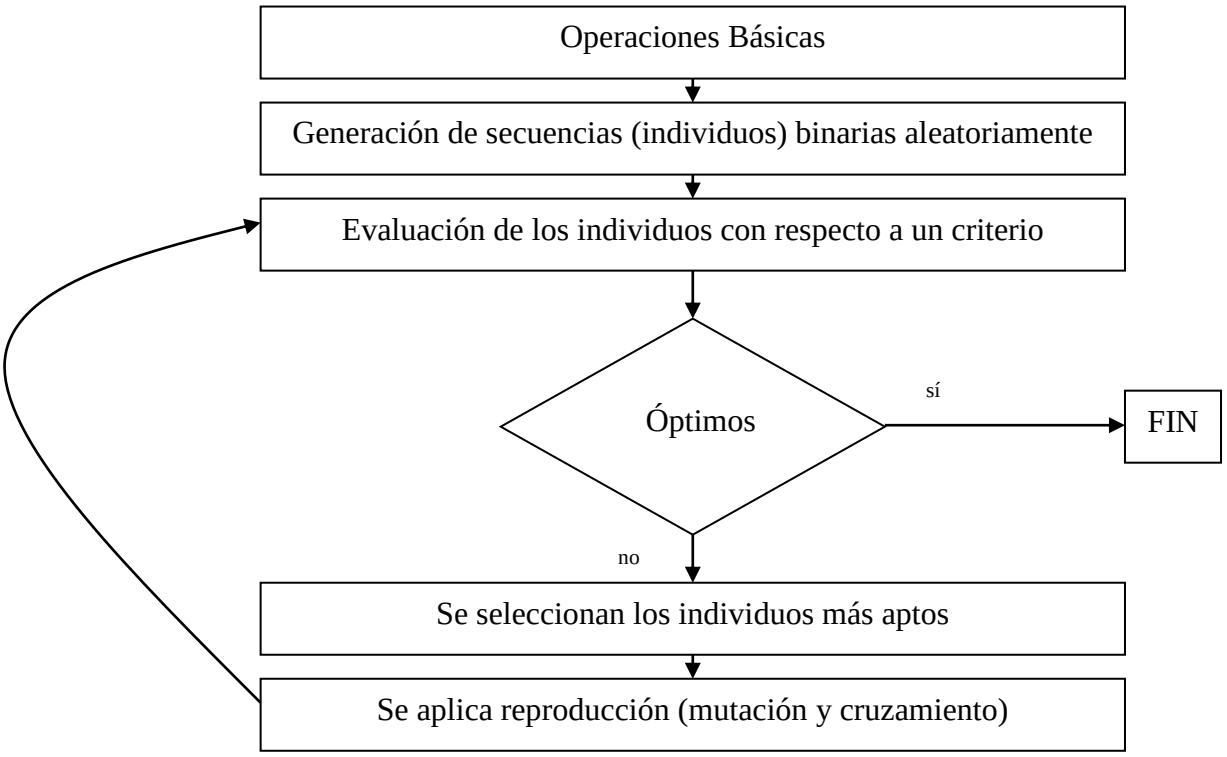
Adquisición de Comportamientos Básicos (evolución)

ALGORITMOS GENÉTICOS
Los algoritmos genéticos nos permitirán utilizar el concepto de evolución para:
- Afinación de constantes
- Generación de algoritmos de máquina de estado

Adquisición de Conocimiento Nuevo a partir de Comportamientos Básicos (heredados)

REDES NEURONALES
Aprendizaje por medio de ejemplos (tú le enseñas qué debe de hacer en determinadas circunstancias).

¿Qué es un algoritmo genético?



Las operaciones básicas sirven para evaluar a los individuos, es decir se genera una medida para evaluar a cada secuencia de bits.

Ejemplo

Individuo 1	1110 1111 1000 0111 0001
Individuo 2	0001 1001 1000 1111 0000
.	
.	
.	
Individuo N	0110 0001 1110 0010 1111

Ahora suponer que se cruza el individuo 1 con el 2 y tiene dos descendientes:

Individuo 1 ⊗ Individuo 2	1110 **1001** 1000 0111 0001
	0001 **1111** 1000 1111 0000

Para hacerlo "realista" se tienen que tomar partecitas (bloques de bits) del mismo lugar (por ejemplo del segundo bloque).

Mutación:

Individuo 1	1110 1111 **0111** 0111 0001

1. Afinación de Constantes en Campos Potenciales

Se tiene un **cromosoma** que codifica las constantes de un robot que navega usando campos potenciales.

μ	δ	d_1	d_0	η

Por ejemplo si $\mu = 3.5$ y se quiere tener cuatro bits para la parte entera y cuatro para la decimal:

0011 1000
Pues es $(0)2^3 + (0)2^2 + (1)2^1 + (1)2^0 + (1)2^{-1}$
(verificar que esto es cierto)

Si se tienen *n* individuos y se evalúan en un medio ambiente, la función podría ser:

$$f(ind_i) = \frac{k_1}{d(x_*, x_f)} + \frac{k_2}{pasos} + ... + k_n choques$$

En cada generación se cambia el origen y el destino.

En la última generación ya se tienen las constantes adecuadas.

2. Generación de algoritmos de máquinas de estado

Cromosoma que codifica la máquina de estados

# de estados	Estado 1	Estado 2	…	Estado N

Cada estado, a su vez, es un conjunto de variables de salida y decisiones (condicionales – "if"s) con las variables de entradas.

Por ejemplo, el estado 1 se codificaría de la siguiente manera:

# de Variables de Salida	Variable 1	Valor	Variable 2	Valor	…	Variable N	Valor	# Variables Condicionales	Variable a sensar 1	Cota inferior	Cota superior	Transición (estado receptor)

…sigue…

Variable a sensar 2	Cota inferior	Cota superior	Transición (estado receptor)	…	Variable a sensar M	Cota inferior	Cota superior	Transición (estado receptor)	Transición si falso

Ejemplo
Codificar el estado (1) en:

(1) Var5 = 7 → (2)

La solución es el binario de:

1	5	7	0	2

O sea:

001	101	111	000	010

Ojo: necesito una máquina que realmente evalúe este cromosoma y lo ejecute como una máquina de estados.

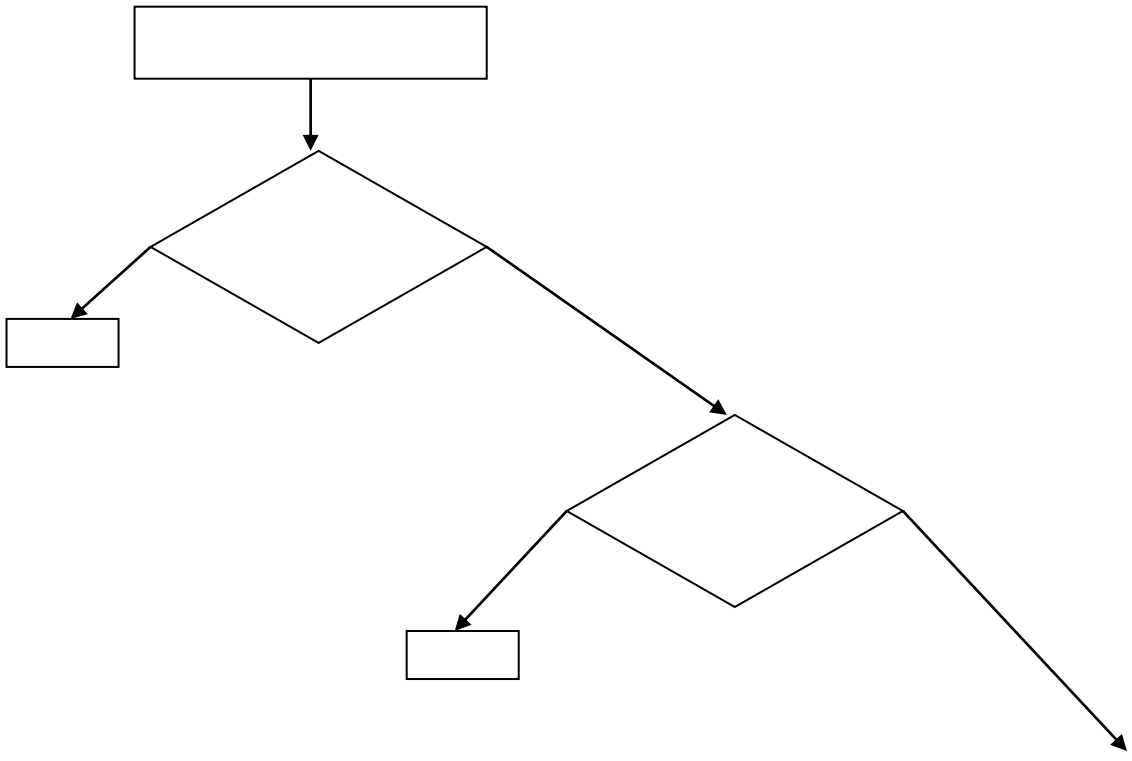
Ejemplo
Codificar la máquina:

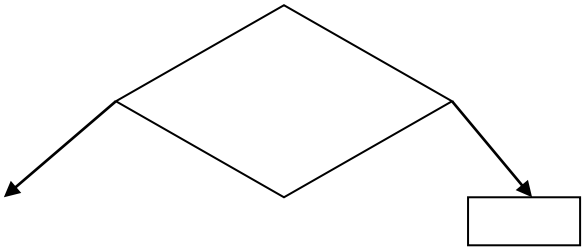
(1) var1 = 3, var4 = 8 → 10 < S3 < 25 ? (2) | (18)

Solución: el binario de:

3	2	1	3	4	8	1	3	10	25	2	18

Ejemplo
Codificar lo siguiente:





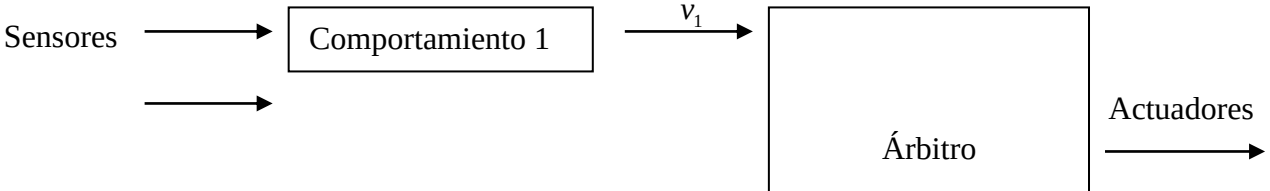
NOTA: LO IMPORTANTE ES QUE EL CROMOSOMA SE PUEDA EJECUTAR. Por ejemplo, si hay condicionales de ambos lados no se puede con esta estructura.

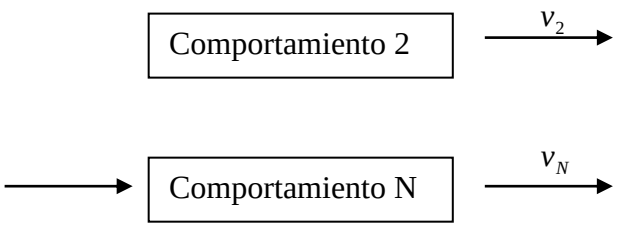
NOTA: Al hacer mutaciones o cruzamientos, hay que tener cuidado. Por ejemplo si se cruzan dos individuos de dos estados, y me da uno de un estado, habría que cortar el hijo para que la información posterior nada más tenga un estado. Igual si cambia a tres, añadir un estado. O cruzar condicionalmente… por ejemplo si cruzo el número de estados, podría tener que cruzar también la información de los estados etcétera.

Ejemplo

Codificar:
(1) M1 = 0.5; M2 = 0.3 → 1.0 < S3 < 1.3 **?**
T: (2) M1 = 0.1, M2 = 0.4 → 3.1 < S2 < 4.1 **?**
T: (2)
F: 2.3 < S5 < 4 **?**
T: (3) M1 = 0.7, M2 = 0.7 → 6 < S2 < 7.0 ?
T: (3)
F: 2.1 < S3 < 4.0 **?**
T: (3)
F: (1)
F: (2)
F: 3.1 < S1 < 4.7 **?**
T: (3)
F: 1.1 < S2 < 1.2 **?**
T: (2)
F: (1)

Utilización de algoritmos genéticos para organizar los comportamientos (arbitraje)





Recuérdese que

$$c_f = \sum_{i=1}^{N} g_i \mathbf{b}_i$$

donde g_i son las constantes de ganancia y $\mathbf{b}_i$ son los vectores de movimiento por los diferentes comportamientos.

Además se tiene una función de desempeño global que indica si el robot cumplió el objetivo.

- Una forma es encontrar las constantes g_i con algoritmos genéticos, basándose en la función de desempeño global.
- Otra forma de organizar es poniendo prioridades.

Método de Utilidad Múltiple (Utility Manifold)

Se quiere que el árbitro decida cuál de los comportamientos se debe activar.
Todos los comportamientos dan salidas siempre.
Cada comportamiento dirá su aplicabilidad, dependiendo de:

- los sensores internos
- los sensores externos

Esto es un valor en (0,1)

1. Función de activación del comportamiento

   $f_{H_i}(S_I, S_E, \mathrm{V})$

   donde

   S_I son los sensores internos (batería, odómetro, etc.)

   S_E son los sensores externos (sonares, infrarrojo, etc.)

   V son variables de estado $x_1, ..., x_r$

   Se modela:

   $f_{B_i}(S_E, S_I, x) = a_{i0} + a_{i1}S_E + a_{i2}S_I + a_{i3}x_i + a_{i4}S_E^2 + a_{i5}S_I^2 + a_{i6}x_i^{\,2} + a_{i7}S_E S_I + a_{i8}S_I x_i + a_{i9}S_E x_i$,

   para $i = 1, ..., n$ los n comportamientos

Se quieren obtener a_{ij} con algoritmos genéticos.

Se definen:

$$x_i = \begin{cases} b_{i1} + b_{i2}\mathbf{e}^{-|b_{i3}|t_i}, & B_i \text{ está activo} \\ 0, & \text{e.o.c.} \end{cases}$$

Para cada comportamiento se tiene una función de activación. Se tienen diferentes individuos y, dentro del cromosoma, se tiene:

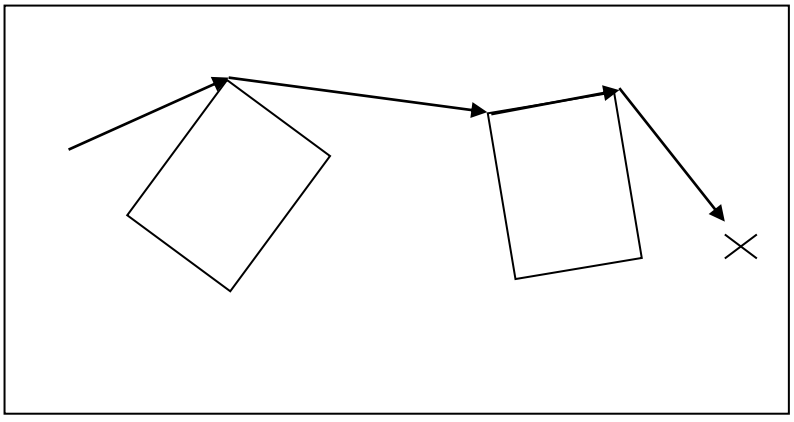
a_{i0}	a_{i1}	…	a_{i9}	b_{i1}	b_{i2}	b_{i3}

(en binario).

Resumen
Los algoritmos genéticos se utilizan para introducir a los robots los comportamientos básicos. Esto NO corre en línea, es para "alambrar" los "instintos" al robot.

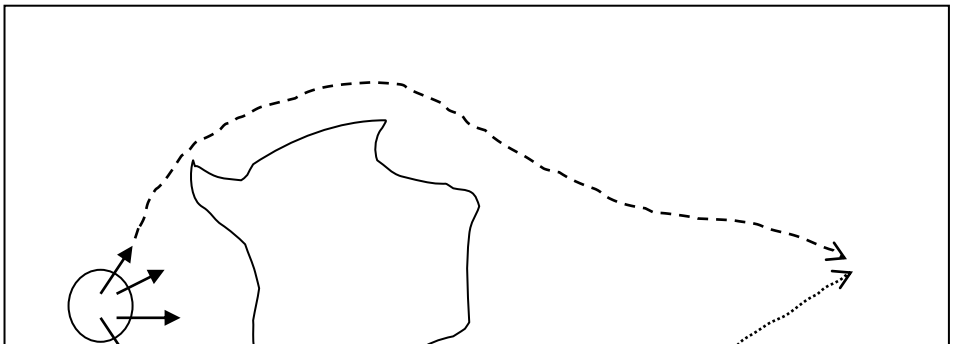
Planeación

¿Cómo planear la ruta de un punto a otro?



Se quisiera encontrar camino más rápido.

Supóngase que se tiene un campo potencial y un obstáculo
¿Cómo se podría ajustar los campos potenciales?:



Una idea podría ser tratar de encontrar *varios* caminos por campos potenciales, dependiendo de un cierto ángulo de desvío θ.

Esto genera un árbol de opciones (caminos) que es muy costoso recorrer con fuerza bruta. ¿Cómo recorremos este árbol para que sea inteligente?

El objetivo es encontrar una mejor ruta para recorrerla.

Búsqueda de la mejor ruta

El problema de búsqueda se puede formular como sigue:

Dado

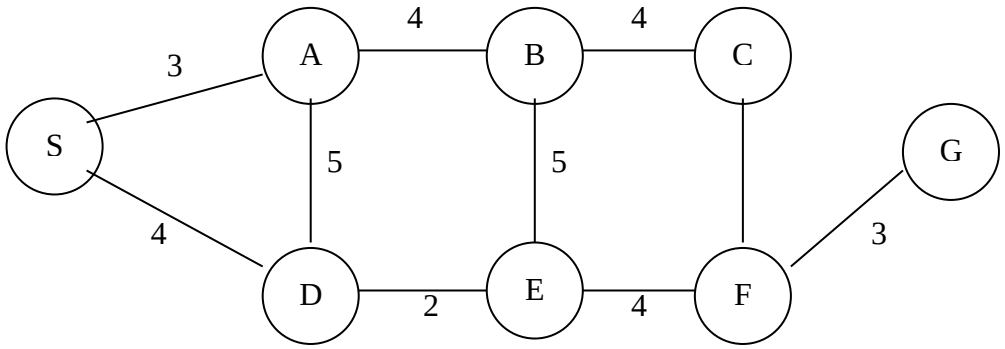
x_0 un punto inicial (o nodo)

x_f un punto objetivo (o nodo)

un grafo con pesos (red topológica)

se quiere encontrar una ruta óptima que llegue al punto objetivo desde el punto inicial y recorrerla.

Ejemplo



Depth-first
Funciona para árboles

Es necesario convertir la red en árbol. Se ponen únicamente los caminos únicos (no recursivos).
Se ponen los pesos acumulados en cada nivel

Este approach se vuelve gigantesco, por lo que es necesario hacer un approach inteligente:

- Se va a realizar la búsqueda recorriendo el árbol por preorden hasta que sea nulo (en cuyo caso se saca de la lista y se mete otro hijo del padre) o sea el objetivo.

S
A|S
B|A|S
C|B|A|S
E|B|A|S
D|E|B|A|S
F|E|B|A|S
G|F|E|B|A|S

Aquí ya se encontró el destino por lo que se encontró un camino de nodos. Ésta es la forma de encontrar **un** camino.

La aplicación es:

1) El planeador encuentra los nodos del grafo que hay que visitar
2) Cada uno se convierte en un punto de atracción sucesivo en el algoritmo de campos potenciales

Hill Climbing

Tiene una función heurística.
Generalmente la función del costo (o peso) al nodo más la distancia remanente al destino
$f\left(d\left(q_j, q_F\right), w_{ij}\right)$
Es el costo de pasar de un nodo a otro más (por ejemplo) la distancia euclideana en R^3 del punto físico asociado al nodo y el punto físico asociado al destino.

Breadth-Search

Se va guardando en una lista ligada
En cada iteración se van cargando los hijos de cada uno de los nodos de cada nivel.

Planeación usando espacio-estado

Reglas nombre {
  Precondiciones
  →
  Operadores
  Lista de Borrado
  Lista Aditiva
}

Por ejemplo: Ponerse los zapatos:

Regla Colocar_Calcetin_Derecho {
  Precondiciones
    Colocar Zapato Derecho (comando)
    Pie derecho está desnudo
    Calcetín está disponible
  →
  Operador
    Poner_Calcetin_PieDerecho
  Lista de Borrado
    Calcetín está disponible
    Pie derecho está desnudo
}

Regla Colocar_Zapato_Derecho {
  Precondiciones
    No hay pie derecho desnudo
    Colocar Zapato Derecho (comando)
    Zapato derecho disponible
  →
  Operador
    Poner zapato derecho
  Lista de borrado
    Zapato derecho disponible
    Colocar zapato derecho
}

Se pueden organizar jerárquicamente:

Inicio
  a) Colocar_Calcetin_derecho
      a. Colocar_zapato_derecho
          i. Colocar_calcetin_izquierdo
          ii. Colocar_zapato_izquierdo
      b. Colocar_calcetin_izquierdo
          i. Colocar_zapato_izquierdo
              1. Colocar_zapato_derecho
          ii. Colocar zapato derecho
              1. Colocar zapato_izquierdo
  b) Colocar_Calcetin_izquierdo
      a. Colocar zapato izquierdo

i. Colocar calcetín derecho
ii. Colocar zapato derecho

b. Colocar calcetín derecho
i. Colocar zapato derecho
1. Colocar zapato izquierdo
ii. Colocar zapato izquierdo
1. Colocar zapato derecho

¿Por cuál camino irnos?
Debemos ponerle pesos y después trabajarlo con un algoritmo de búsqueda.

Ejemplo

Bloques

B E
A C D

Se quiere tener una representación espacio-estado con los operadores.
Por ejemplo:

C
B E
D A

Operadores:

Goto(x,y,z)	Ir a (x,y,z)
Pickup(w)	Recoger w de la mesa
Putdown(w)	Soltar w en la mesa
Stack(u,v)	Poner el u encima del v
Unstack(u,v)	Quita el bloque u del v (y se lo queda en la mano)

La representación del estado del mundo:

Location(w,x,y,z)	el bloque w está en las coordenadas x,y,z
On(x,y)	el bloque x está encima de y
Clear(x)	no hay algo encima de x
Gripping(x)	el brazo del robot tiene el bloque x (gripping(*void*) es que no tiene algo)
OnTable(w)	el bloque w está en la mesa

A partir de esto se deduce que

$$\forall x\left(CLEAR(x) \Rightarrow \neg\exists y\left(on(y,x)\right)\right)$$

$$\forall x\left(GRIPPING(void) \Leftrightarrow \neg GRIPPING(x)\right)$$

$$\forall y \forall x \left(ONTABLE(y) \Rightarrow \neg ON(y, x) \right)$$

Estado 1

OnTable(A)
OnTable(C)
OnTable(D)
On(B,A)
On(E,D)
Gripping(void)
Clear(B)
Clear(C)
Clear(E)

Regla Pickup(x) {
  Precondiciones
    Clear(x)
    OnTable(x)
    Gripping(void)

  →
    Lista de adición
      Gripping(x)
    Lista de borrado
      OnTable(x)
      Gripping(void)
}

Observación: esto genera una *notación de predicados*
$\forall x\big(PICKUP(x) \Rightarrow GRIPPING(x)\big) \leftarrow \big(GRIPPING(void) \wedge CLEAR(x) \wedge ONTABLE(x)\big)$

Regla PutDown(x) {
  Precondiciones
    Gripping(x)
  →
  Lista Adición
    Gripping(void)
    OnTable(x)
  Lista de Borrado
    Gripping(x)
}

Regla Stack(x,y) {
  Precondiciones
    Gripping(x)
    Clear(y)
  →
  Lista de adición
    Gripping(void)
    On(x,y)
  Lista de Borrado
    Gripping(x)
    Clear(y)
}

Árbol de Estados

  B E
  A C D

1) B E
A C D

2) B E
A C D

Etcétera (todas las posibilidades)

Tercera evaluación
Comportamiento de potenciales
Comportamiento de máquinas de estado
Árbitro

(con variables de aplicabilidad entre 0 y 1)

Máquina de Inferencias

Hechos + Reglas → Máquinas de Inferencia → Agenda
Los hechos activan reglas. Las reglas pueden generar otros hechos que, a su vez, pueden activar nuevas reglas.

Por ejemplo esto puede evitar que en 100000 if's anidados se tenga que ejecutar el último.
En lugar de que las reglas busquen los hechos que los activan, es al revés (puesto que los hechos generalmente no cambian).

Las reglas producen hechos NUEVOS.

El sistema **CLIPS** es un lenguaje de programación orientado a plantillas que permite definir sistemas expertos. Tiene una máquina de inferencia con encadenamiento hacia delante para eficiencia.
Permite definir hechos, reglas, etc.

Tercer tarea
Incorporar campos potenciales de obstáculos desconocidos.
Mapa topológico fijo (la red topológica es fija y predeterminada) Moverse entre nodos con campos potenciales. Asignar el destino final como el nodo más cercano al destino. Buscar en la red topológica correspondiente para un camino óptimo (Dijkstra, etc.) bajándolo de la red.

1. planeador global
2. comportamientos: campos potenciales
3. comportamientos: máquina de estado

Lenguaje Natural

La idea es hablarle a un robot en lenguaje cotidiano y que nos "entienda". Entender es hacer un análisis semántico para obtener un significado y transformarlo en una acción.

Lo que yo le dije, que lo haga.

La definición operacional de **entender** es que el sistema realice las acciones que el usuario le pide.

Entender algo es transformar lo que se pide en una **representación**, la cual ha sido escogida para que corresponda a un conjunto de acciones disponibles que pueden ser ejecutadas.

1. Señal de entrada
   Puede ser voz, imagen o combinación de ambas (o cualquier otra forma de entrada, por teclado).
   Voz: por medio de un micrófono que transforma una señal acústica en una señal eléctrica.
   Es necesario aplicar técnicas de procesamiento digital de señales para obtener las características de esa señal acústica.
   Existen algoritmos (como el de cuantización vectorial y sus variantes) para procesar e identificar las palabras.

2. Las palabras de entrada son revisadas para ver si ellas están agrupadas de acuerdo con reglas gramaticales, significando que ellas forman oraciones gramaticalmente correctas.

3. El significado de cada palabra y de la oración es asignado.
   Este paso es el más difícil de los tres.

Dependencia Conceptual

Primitivas Conceptuales

La representación de varias acciones es la misma.
Por ejemplo: "Juan dio el libro a Pedro", "Juan prestó el libro a Pedro", "Juan regaló el libro a Pedro", significan cosas diferentes, pero la acción de transferir el libro de una persona a otra es la misma.

Cada primitiva tiene:
a) Un actor (realiza el acto)
b) Un acto (realizado por el actor y hecho hacia un objeto)
c) Un objeto (ente sobre el que es realizada la acción)
d) Una dirección (en la cual el acto es realizado)
e) Un estado (en el cual el objeto se encuentra)

Las primitivas son:

PTRANS Es la transferencia física de un objeto.
(PTRANS (Actor NIL) (Objeto NIL) (FROM NIL) (TO NIL))

Por ejemplo

"Robot, lleva las flores al patio"
Se representaría como:
(PTRANS (Actor Robot) (Objeto flores) (From Posicion_flores) (To Patio))

"Robot, ve a la cocina"
(PTRANS (Actor Robot) (Objeto Robot) (From Posicion_robot) (To Cocina))

Notar que el verbo indica qué primitiva utilizar

MTRANS es la transferencia de información mental.
Roberto le dijo a Susana que era bonita
(MTRANS (ACTOR Roberto) (OBJETO "que Susana es bonita") (FROM cerebro_roberto) (TO cerebro_susana)

INGEST es la ingestión de un objeto por un animal, actor o ente
Juan toma leche
(INGEST (ACTOR Juan) (OBJECT Leche) (FROM NIL) (TO Boca_Juan))

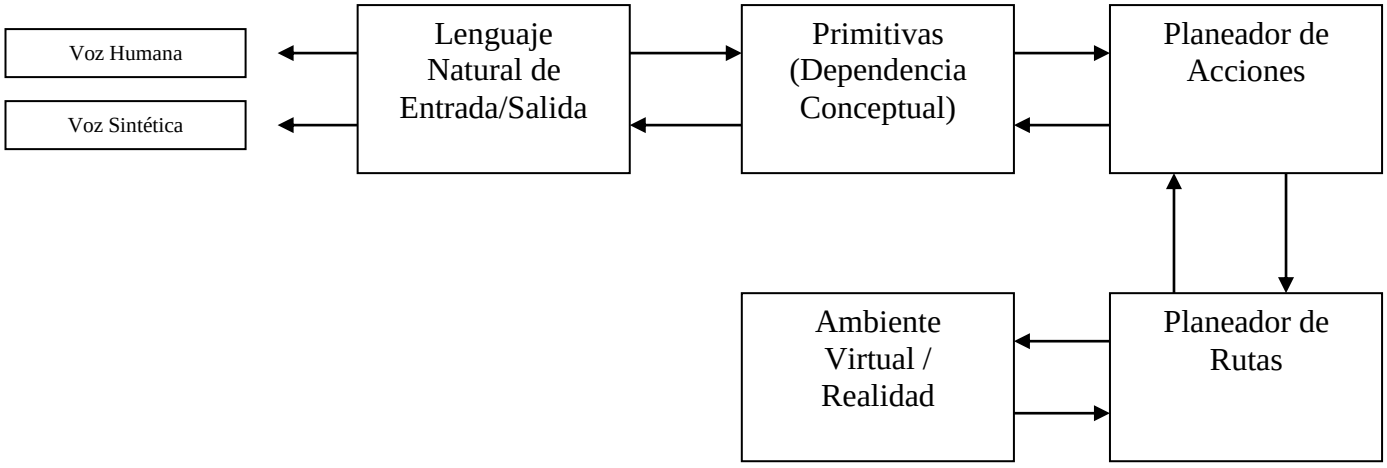
PROPEL es la aplicación de fuerza física a un objeto

María mató a una araña aventándole un zapato
(PROPEL (ACTOR María) (OBJECT Zapato) (FROM Brazo_Maria) (TO Araña))

Observar: para matar a la araña con el PROPEL, primero tuvo que mover el brazo.

MOVE es el movimiento de una parte del cuerpo

Voz → Lenguaje natural de entrada y salida → PRIMITIVAS (Dependencia Conceptual) → Planeador de acciones → Planeador de Rutas → Ambiente Virtual / Realidad



Proyecto Final

1) Red topológica con nodos
2) Encontrar y ejecutar la mejor ruta
3) 6 cubos en un array de 2x3

A F |
B E |
C D |
Cuarto 1 cuarto 2

Poder ejecutar sentencias tipo “lleva A al cuarto 2”
(por ejemplo el A no se puede agarrar, hay que quitar C y B y ponerlos en otro lado)

Con CLIPS
Podemos poner nodos topológicos.

Creación de Mapas

La idea básica es crear un modelo del medio ambiente (una red topológica), basándose en la información en los sensores.
Detectar con los sensores los puntos que delimitan el "espacio libre" (es decir, por donde puede navegar el robot).

Diagramas de Voronoi

La idea es encontrar el diagrama de Voronoi de los puntos y entones el robot puede caminar sobre las líneas.

Considérese un punto $(x, y) \in C$ (el espacio libre). Los puntos bases de (x, y) son los puntos (x', y') más cercanos en el espacio ocupado $\bar{C}$. El diagrama de Voronoi es el conjunto de puntos en el espacio libre que tiene cuando menos dos puntos bases diferentes a la misma distancia.

Puntos Críticos
Los puntos críticos (x, y) son puntos en el diagrama de Voronoi tales que minimizan un margen local.
Existe $\varepsilon > 0$ tal que el margen de todos los puntos en la bola abierta de (x, y) no es menor.

Líneas críticas
Son obtenidas conectando cada punto crítico con sus puntos base. Las líneas críticas particionan el espacio libre en regiones disjuntas.

Cuantización Vectorial

Hacer clustering sobre el espacio libre.
Clustering:
Considérense m vectores, $v_i = (x_i, y_i)$.

1. Encontrar un centroide $c_1 = \frac{1}{m}\sum_{j=1}^{m} v_j$
2. Generar dos centroides nuevos a partir del centroide c_i anterior

$$c_{i+1} = c_i + \varepsilon_1$$
$$c_{i+2} = c_i \varepsilon_2$$

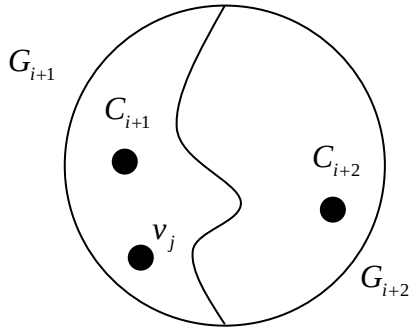
Donde $\varepsilon_k = 1 + \delta_k$, con δ_k pequeño en magnitud absoluta

3. Agrupar los vectores en dos grupos de acuerdo al centroide más cercano.

$$d_r = d(v_j, c_{i+r}) \quad r = 1, 2$$

Asignar a los grupos:

$$v_j \in G_{i+1} \Leftrightarrow d_1 < d_2$$
$$v_j \in G_{i+2} \Leftrightarrow d_1 \geq d_2$$



Localización del Robot

Si el odómetro fuera perfecto, se puede saber siempre en dónde se encuentra el robot. Sin embargo, en la realidad no son exactos y no se sabe.

Se puede utilizar la visión para localizar **puntos de referencia** predefinidos.

Para hacer esto, se necesita que el sistema de visión sea invariante al tamaño y a transformaciones (rotaciones).

Ya sea con un sistema de visión o de localización, se puede hacer triangulación.

Si se tienen varias marcas, suponiendo que se pueden encontrar las distancias a las marcas, se puede trazar un círculo de radio la distancia.

Se establece entonces:

$$(z - z_i)^2 + (x - x_i)^2 + (y - y_i)^2 = d_i^2 \qquad \forall i$$

El sistema de ecuaciones me determina (x, y, z), es decir, la posición del robot.

Esto podría funcionar pero los sensores tienen errores y por lo tanto nunca se intersecan los círculos.

Por lo tanto, la localización del robot es probabilística.

4. Si $\left(D_g^t - D_g^{t-1}\right) > \varepsilon$ entonces recalcular los centroides

$$c_{i+r} = \frac{1}{m_{G_{i+r}}} \sum_{j=1}^{m_{G_{i+r}}} v_j, \ r = 1,2$$

donde $D_g^t = \sum_{j=1}^{m} \min\left(d_1(j), d_2(j)\right)$ y t es la iteración

Si es mayor, ir al punto 3.
Si es menor:

Repetir 2 hasta que se tenga el número de centroides deseado.

Cadenas de Markov y Modelos Ocultos de Cadenas de Markov

Una cadena de Markov es como una máquina de estado en la que las transiciones son probabilísticas.

$$a_{ij} = P\left[\text{pasar de i a j}\right]$$

Los cambios de estado están indexados al tiempo y se denominan q_t.

Un proceso estocástico es un **proceso markoviano de primer orden** si la probabilidad de que la cadena de Markov se encuentre en un determinado estado j es:

$$P\left[q_t = j \mid q_{t-1} = i_1, q_{t-2} = i_2, ..., q_0 = i_t\right] = P\left[q_t = j \mid q_{t-1} = i\right]$$

Se tiene:

$$a_{ij} > 0 \ \ \forall i, j$$
$$\sum_{j=1}^{N} a_{ij} = 1 \ \ \forall i$$

<falta una clase aquí>

Problema #1

$O = \left(O_1, O_2, ..., O_N\right)$
$\lambda = \left(A, B, \Pi\right)$
$P\left(O \mid \lambda\right)$

Con lo visto anteriormente, son aproximadamente $(2T-1)N^T + N^T - 1$ operaciones. Por ejemplo, con 5 estados y 100 observaciones, esto es aproximadamente 10^{72} operaciones. Por tanto, necesitamos obtener algún procedimiento más eficiente.

Procedimiento hacia delante.

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = i \mid \lambda)$$

Es la probabilidad de observar la secuencia parcial $O_1, O_2, ..., O_t$ y el estado *i* en el tiempo *t*.

Esta probabilidad se puede encontrar de manera inductiva:

1) Inicialización

$$\alpha_1(i) = \Pi_i b_i(O_1), \ 1 \le i \le N$$

O_i es la observación al tiempo *i*.
Por ejemplo, se puede tener: $O_1 = v_4$ que es una mesa.

Las probabilidades son fijas pero las observaciones están cambiando respecto al tiempo.

2) Inducción

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \quad 1 \le t \le T-1, \ 1 \le j \le N$$

La probabilidad depende de la probabilidad anterior, las probabilidades de transición y la probabilidad de la observación en el estado *j*.

3) Término

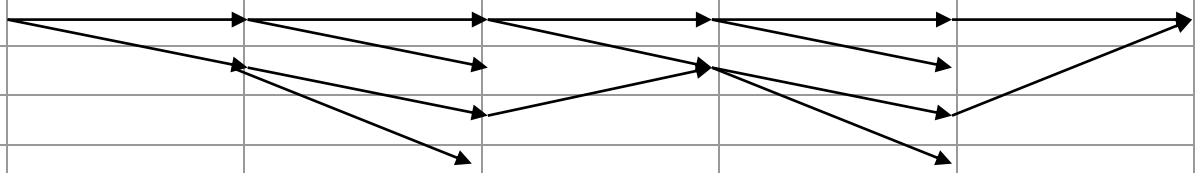
$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{k=1}^{N} P(O_1, O_2, ..., O_T, q_T = i \mid \lambda)$$

Para calcular $\alpha_t(j) \quad 1 \le t \le T, 1 \le j \le N$ se requieren N^2T cálculos (en lugar de $2TN^T$)

Problema #2
Encontrar la mejor secuencia de estados $q = (q_1, q_2, ..., q_T)$ dada la secuencia de observaciones $O = (O_1, O_2, ..., O_T)$
En términos del robot: dado lo que vio, en dónde se estuve moviendo.

Diagrama de Trellis
Dado una máquina de estados con transiciones:

	Tiempo				
Estado	0	1	2	3	4
S_0					
S_1					
S_2					
S_3					



Para cada estado hay una serie de caminos, hay que encontrar el mejor camino. Sin embargo, para cada tiempo, la búsqueda crece en forma exponencial (el número de caminos crece)

Hay que hacerlo con el **Algoritmo de Viterbi**, que va eliminando todos los caminos menos los más probables.

Algoritmo de Viterbi
Sea $\delta_t(i)$ la mejor ruta en el tiempo *t* que toma en cuenta las primeras *t* observaciones y termina en el estado *i*.

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P\left[q_1 q_2 \ldots q_{t-1}, q_t = i, O_1, O_2, \ldots, O_t \mid \lambda\right]$$

1) Inicialización

$\delta_1(i) = \Pi_i b_i(O_1)$ para toda *i*.

2) Recursión

$$\delta_t(j) = \left(\max_{1\le i\le N}\left[\delta_{t-1}(i)\, a_{ij}\right]\right) b_j(O_t)$$
$$2 \le t \le N,\ 1 \le j \le N$$
$$\Psi_t(j) = \arg\max_{1\le i\le N}\left[\delta_{t-1}(i)\, a_{ij}\right]$$ (es decir, los nodos que conforman la ruta)

3) Término

$$p^* = \max_{1\le i\le N}\left[\delta_T(i)\right]$$
$$q_T^* = \arg\max_{1\le i\le N}\left[\delta_T(i)\right]$$

4) Ruta
La ruta se obtiene:

$$q_t^* = \Psi_{t+1}\left(q_{t+1}^*\right), \quad t = T-1, T-2, ..., 1$$

Problema 3
Estimación de Parámetros

Se quiere encontrar $\lambda = (A, B, \Pi)$ que satisfaga un cierto criterio de optimalidad.

Modelo inicial → (de datos de entrenamiento) Segmentación secuencia de estados → Reestimación del modelo → Convergencia del modelo? Si sí, regresar a segmentación, si no, dar parámetros del modelo.

$b_j(k)$ es el número de observaciones con el símbolo *k* en el estado *j* dividido por el número de observaciones en el estado *j*.

a_{ij} es el número de transiciones del estado *i* al estado *j* dividido por el número de transiciones de *i* a cualquier estado.

Π_i es el número de veces que se empezó en el estado *i* dividido entre el número de veces de entrenamientos.