

# Instituto de Computación

## Facultad de Ingeniería

# Acercamiento a los Robots Inteligentes

Ricardo J. Bagnasco

**Proyecto de Grado, 2002**

**Tutores**: Gonzalo Tejera y Carlos Luna

Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay.

# Indice

# Indice de Figuras y Tablas

# Abstract

En los últimos años han comenzado a aparecer con una frecuencia cada vez mayor noticias sobre robots y sus aplicaciones, ya no como ciencia ficción sino como una realidad tangible. Esto hace suponer que estamos alcanzando la madurez necesaria en cuanto a las técnicas para construir aplicaciones reales. En este trabajo presentamos los resultados de la investigación del estado del arte de la robótica en tres ámbitos: las noticias, las universidades, y la industria. Además mostramos diferentes técnicas para programar a un robot, y elegimos una (*q-learning*) para resolver dos problemas diferentes: 1º seguir paredes; y 2º empujar un objeto hasta llevarlo a una posición conocida. Explicamos así mismo el proceso de investigación y selección de las herramientas de hardware y software consideradas al momento de construir las soluciones para los problemas planteados.


**Palabras claves:** robótica, robot autónomo, Q-learning, aprendizaje basado en comportamientos, mindstorm de lego.

# 1. Temas Introductorios

## *1.1 Reflexiones iniciales*

La robótica es un campo realmente apasionante para muchos de nosotros; durante años libros y películas de ciencia ficción nos han hecho soñar con diferentes futuros posibles donde coexistamos con robots. En la actualidad lamentablemente nos encontramos bastante lejos de lograr lo que nos presentan las películas. Para averiguar cuán alejados estamos es que realizamos un estudio del estado del arte de los robots comerciales, los cuales cumplen principalmente la función de entretener sin realizar un trabajo "útil", aún cuando pueden utilizar tecnología avanzada.

Consideramos que el siguiente paso de la computación una vez que se han desarrollado numerosas aplicaciones para computadoras personales y mainframes, es dar lugar a la programación de aplicaciones no tan "virtuales", que funcionen dentro de un computador, sino más "reales" o físicas que se desempeñen en el mundo real. Creemos que tendríamos que pasar a construir aplicaciones que tuvieran un componente más dinámico y autónomo. Igualmente pensamos que el salto cualitativo sólo es posible al programar agentes que siendo autónomos resuelvan una tarea de forma no supervisada.
Como futuros ingenieros, queremos ser constructores de este porvenir, para ello comenzamos encarando la labor de programar a un agente autónomo para realizar tareas, con las herramientas y técnicas hoy disponibles. En el camino nos encontramos con algunas peculiaridades propias de dicha tarea, éstas difieren de todo lo que hemos visto con anterioridad en las aplicaciones efectuadas a lo largo de la formación como Ingenieros en la Facultad. Hay que tener en cuenta que un robot autónomo es una máquina que opera en ambientes parcialmente desconocidos, cambiantes e impredecibles. Estas características, hacen difícil emplear los métodos tradicionales de la ingeniería basados en análisis del problema, modelación y diseño.

En esta tesis se presentará un estudio del estado del arte referido a robots industriales por tratarse del grupo de robots de mayor difusión y uso en la actualidad; también se hará una recorrida por proyectos académicos, con el fin de conocer el estado actual de la investigación en otras universidades del mundo. También incluimos un par de aplicaciones desarrolladas por nosotros mismos donde expondremos algunas de las características particulares de la tarea de programar una aplicación con un robot y como fueron afrontadas.

Nos hemos esforzado para que esta tesis sea completa en sí misma, por lo que continuaremos explicando donde se enmarca el tema de nuestro proyecto de grado: la robótica, intentando diferenciar y aclarar conceptos que suelen mezclarse; además, se comentan otros temas necesarios para una mejor comprensión del proyecto (si ya se poseen conocimientos sobre estos temas puede saltearse esa parte). En el capítulo 2 se presentará un estudio sobre el estado del arte de la robótica, dividido en tres áreas: los *robots industriales*; *robots comerciales* y *prototipos*; y *proyectos académicos de robótica* en diversas universidades del mundo. Sería más exacto decir universidades *europeas* y *norteamericanas* en lugar de universidades *del mundo*; pero en los hechos son éstas las que marcan la tendencia del resto del mundo académico –al menos en el área de la robótica- por lo que nos pareció suficiente estudiar qué se esta haciendo en ellas, ya que las demás universidades se limitan lamentablemente a seguir el camino indicado por las mismas. En el capítulo 3 se presentarán los aspectos más destacables de las aplicaciones que implementamos. Y para finalizar en el capítulo 4, las conclusiones del proyecto. Se incluye luego un glosario para evitar ambigüedades en la terminología usada a lo largo de esta tesis, ya que no existe un vocabulario preciso universalmente aceptado en el área de la robótica y no resulta raro encontrar variaciones en el significado de las palabras en la literatura sobre el tema. Recientemente ISO (siglas en inglés para la Organización Internacional para Estándares) y organizaciones de estándares nacionales han comenzado a definir una terminología para la robótica ya que hasta el momento se carecía de un marco importante ya que permitiera manejar un lenguaje y unidades de medida común.

## *1.2 Acercamiento a la Inteligencia Artificial y a la Robótica*

En esta sección se destacará el área de la Inteligencia Artificial dentro de la cual se encuentra la robótica; se explicará qué es un robot y se definirá la arquitectura de uno; se desarrollará el tema de sensores y accionadores (componentes fundamentales de un robot inteligente); posteriormente se comentaran algunos enfoques utilizados a la hora de construir aplicaciones con robots.

### *1.2.1 Inteligencia Artificial*

(basado en el trabajo de John Mc Carthy de la Universidad de Stanford, EE.UU.
[*http://www-formal.stanford.edu/jmc/whatisai/whatisai.html*] )

Se puede entender a la Inteligencia Artificial (IA) como la ciencia e ingeniería de hacer máquinas inteligentes, más específicamente, programas de computadoras inteligentes. La IA está relacionada con la tarea de usar las computadoras para comprender la inteligencia humana, pero no está limitada a métodos biológicamente observables.

Esto nos lleva a tener que proporcionar una definición sobre lo que es para nosotros la inteligencia en este contexto. Inteligencia es la parte computacional de la habilidad para lograr metas en el mundo. Igualmente hay problemas para delimitar qué procedimientos se consideran inteligentes y cuáles no, debido a que nuestra comprensión sobre los mecanismos de la inteligencia es limitada.

Alan Turing en un artículo que apareció en 1950 (*Computing Machinery and Intelligence*, disponible en Internet [*http://www.abelard.org/turpap/turpap.htm*]) discute las condiciones para considerar a una máquina como inteligente. Plantea que si la máquina puede fingir ser un humano ante un observador experto, entonces debiera ser considerada inteligente. Ese test podrá satisfacer a la mayoría de las personas pero nunca a los filósofos. La idea del test es que el observador puede interactuar con la máquina y con otro humano mediante un teclado y una pantalla (para evitar el tema de tener que simular la voz humana como parte del problema), y el humano debe convencer al observador de que él es el humano y la máquina tratará de engañar al observador para que piense que ella es en realidad el humano. Hay que observar que el test de Turing es condición suficiente pero no necesaria, es decir que una máquina que pase el test debiera considerarse inteligente, pero una máquina podría aún considerarse inteligente sin saber lo suficiente sobre los humanos como para imitarlos. Existen varias discusiones sobre este test, por ejemplo Daniel Dennett en su libro *Brainchildren* presenta diferentes versiones parciales del test con restricciones y observa que algunas personas son fácilmente inducidas a creer que un programa más bien tonto es inteligente.

También se ha planteado la cuestión de si las computadoras son el tipo de máquina adecuado para hacer inteligente. La respuesta más simple es que las computadoras pueden programarse para simular cualquier tipo de máquina. Muchos investigadores inventaron máquinas no-computadoras, con la esperanza de que pudieran ser inteligentes en formas diferentes a la que los programas de computadora pueden serlo. Sin embargo, usualmente simulaban sus máquinas inventadas en una computadora y dudaban si valía la pena construirlas. Debido a que se han gastado varios millones de dólares para hacer a las computadoras cada vez más veloces, otro tipo de máquina debiera ser muy rápida para desempeñarse mejor que un programa en una computadora simulando a dicha máquina.

Otra posibilidad planteada habitualmente es la de construir una máquina niño que pudiera mejorarse leyendo y aprendiendo de la experiencia. Esta idea ha sido propuesta varias veces desde la década del 1940 y tal vez podría funcionar. Sin embargo, hoy por hoy los programas de IA no han alcanzado el nivel de ser capaces de aprender lo que un

niño aprende de la experiencia física. Ni comprenden el lenguaje natural lo suficientemente bien como para aprender mucho leyendo.

Tenemos ejemplos supuestamente exitosos como el caso del ajedrez, donde DeepBlue venció a Kasparov, por lo que algunos piensan que ya hemos alcanzado un nivel importante en el desarrollo de la IA. Esto es incorrecto, porque los programas que juegan al ajedrez tienen unos pocos y limitados mecanismos comparados con los usados por los ajedrecistas humanos, y substituyen esa carencia con grandes cantidades de cómputo (además se cree que en parte DeepBlue fue programada para vencer a Kasparov estudiando su técnica y movimientos). Una vez que comprendamos mejor estos mecanismos, podremos construir programas que jueguen al nivel de los humanos con muchísima menos computación a la usada actualmente. Por ejemplo alcanza con ver que la computadora DeepFritz que calcula 3.5 millones de jugadas por segundo comparadas con los 200 millones de DeepBlue, es sin embargo superior a esta porque de nada sirve calcular 200 millones de movimientos si el primero parte de una idea equivocada. DeepFritz evalúa mejor la posición inicial para dictaminar si hay ventaja de las blancas, las negras, o si el juego está igualado. Y por tanto, sus cálculos son más precisos y coherentes. Sin embargo, los aspectos competitivos y comerciales de hacer computadoras que jueguen ajedrez han precedido al uso del mismo en ambientes científicos, y el común de la gente continuará creyendo que este es un indicador de niveles de inteligencia cercanos a la humana.

Existen ciertas corrientes que piensan que jamás se podrá alcanzar el nivel de inteligencia buscado porque habría problemas teóricos acerca de la complejidad computacional que serían claves en el tema de la IA. En los años 30, Kurt Gödel y Alan Turing establecieron la inexistencia de algoritmos que pudieran garantizar la solución de todos los problemas en ciertos dominios matemáticos importantes. Por ejemplo si una sentencia lógica de primer orden es un teorema, o cuándo una ecuación polinomial con varias variables tiene soluciones enteras. Los humanos resolvemos problemas en esas áreas todo el tiempo, y este argumento ha sido ofrecido como razón (con algunas decoraciones) de que las computadoras son intrínsecamente incapaces de hacer lo que las personas hacen. Sin embargo, los individuos no pueden garantizar el resolver problemas a*rbitrarios* en esas áreas tampoco. Otros argumentos contrarios a la inteligencia artificial se apoyan en el trabajo en los años 60 de Steve Cook y Richard Karp, quienes desarrollaron la teoría de los problemas del dominio NP-completo. Los problemas en esos dominios son resolubles, pero en un tiempo exponencial en el tamaño del problema (uno de los problemas NP-completos es *cuáles sentencias del cálculo proposicional son satisfactibles*). Las personas resolvemos a menudo problemas NP-completos en tiempos bastante menores al garantizado por los algoritmos generales, pero no podemos resolverlos muy rápidamente en general por lo que eso no significa que no se pueda alcanzar un nivel aceptable en esta área también. Lo importante para la IA es tener algoritmos tan capaces como las personas al momento de resolver problemas.

## *Ramas de la IA (según la Association for Computing Machinery):*

(esta lista no debe ser tomada como completa)

- Programación automática: Verificación y síntesis
- Razonamiento automático
- Representación del conocimiento
- Metodología de la programación en IA
- Aprendizaje
- Procesamiento del lenguaje natural
- Resolución de problemas, métodos de control y búsqueda
- Robótica
- Interpretación de imágenes y visión artificial
- Inteligencia artificial distribuida

## *Observaciones sobre algunas Aplicaciones de la IA:*

(basado en parte en el trabajo de John Mc Carthy de la Universidad de Stanford, EE.UU.
[*http://www-formal.stanford.edu/jmc/whatisai/whatisai.html*] )

- **Jugar:** se pueden construir máquinas que juegan al ajedrez con un gran nivel por unos pocos cientos de dólares. Si bien hay algo de IA en esto, este tipo de aplicaciones operan principalmente por fuerza bruta, eligiendo el próximo movimiento a realizar buscando entre cientos de miles de posiciones.

- **Reconocimiento de voz:** en los noventa, el reconocimiento de voz alcanzó un nivel práctico aunque para propósitos limitados. Por eso por ejemplo la compañía de aviación norteamericana United Airlines, reemplazó su teclado para información de vuelo por un sistema que usa reconocimiento de voz para números de vuelo y nombres de ciudades. Por otro lado, mientras es posible instruir algunas computadoras usando la voz, la mayoría de los usuarios siguen prefiriendo todavía el teclado y el mouse por ser más convenientes y precisos.

- **Comprensión del lenguaje natural:** para que una computadora comprenda lo que se le está informando no es suficiente con abastecerla con una secuencia de palabras. Tampoco lo es parsear sentencias. La computadora debe ser provista de una comprensión del ámbito sobre lo que versa el texto, y esto es actualmente posible sólo para dominios limitados.

- **Visión por computadora:** el mundo está compuesto por objetos tridimensionales, pero las entradas del ojo humano y las cámaras usadas por computadoras son bidimensionales por lo que hay que elaborar la información para construir una representación adecuada.

- **Sistemas expertos:** se basan en la simulación del razonamiento humano, son intermediarios entre el experto humano, que transmite sus conocimientos al sistema, y el usuario de dicho sistema que lo emplea para resolver los problemas que se le plantean. En el estado actual de la IA, la utilidad de los sistemas expertos depende del sentido común de sus usuarios.

- **Clasificación heurística (datamining):** una de las áreas más factibles para los sistemas expertos -dado el conocimiento actual sobre la IA- es clasificar algún dato en una categoría (de un conjunto fijo de categorías) usando varias fuentes de información. Por ejemplo, aconsejando cuando aceptar una compra con tarjeta de crédito, considerando que contamos con información disponible acerca del propietario de la tarjeta, su registro de pagos y también acerca de los artículos que está comprando e información del establecimiento donde está comprando (por ejemplo acerca de si han ocurridos fraudes con tarjetas de créditos previamente en ese local).

- **La IA en la robótica:** A finales de los setenta se produjo un nuevo giro en el campo de la investigación relacionada con la inteligencia artificial: la aplicación sobre **robots**. Los robots experimentales creados para estos efectos eran máquinas capaces de recibir información procedente del mundo exterior (por ejemplo a través de sensores o cámaras de televisión), así como órdenes de un manipulador humano. Estos primeros robots experimentales eran bastante más inteligentes que los robots industriales, y lo eran porque disponían de un grado mucho mayor de percepción del entorno que los robots empleados en las cadenas de producción. Como todo avance, esto trajo aparejado un nuevo conjunto de problemas y desafíos. Uno de los principales problemas con que se enfrenta la IA aplicada a los robots es el de la visión. Mientras que la información recibida a través de sensores se puede interpretar con relativa facilidad y pasa a formar parte de la descripción del modelo del universo que emplea el robot para tomar decisiones, la percepción de las imágenes captadas y su interpretación correcta es una labor muy compleja. En cuanto a la interpretación de las imágenes captadas mediante cualquier sistema, se ha logrado ya el reconocimiento de formas pre-programadas o conocidas, lo que permite que ciertos robots lleven a cabo operaciones de reubicación de piezas o colocación en su posición correcta a partir de una posición arbitraria. Sin embargo, no se ha logrado aún que el sistema perciba la imagen tomada mediante una cámara y adapte su actuación al nuevo cúmulo de circunstancias que esto implica. Así, por ejemplo, la imagen ofrecida por una cámara de vídeo de las que se emplea en vigilancia y sistemas de seguridad no puede ser interpretada directamente por el ordenador. En la siguiente sección se ampliará más este punto y se introducirán los conceptos necesarios acerca de Sensores y Actuadores. Otros problemas que enfrenta la IA en la robótica son derivados del hecho de que por primera vez, las aplicaciones informáticas comienzan a funcionar en una interacción directa con el mundo en el que vivimos (el mundo "real"). Se pasa de entornos controlados, con parámetros conocidos o al menos caracterizables, a entornos parcialmente observables, mucho más dinámicos, con la muy factible falla de los sensores del robot, y la dificultad de planear acciones (entre otras diferencias). Todo esto hace que se tengan que inventar soluciones específicas para estos problemas nunca antes enfrentados. Esta área ha crecido enormemente, por lo que si bien los problemas de la robótica pueden verse como pertenecientes a un conjunto más general de problemas de la IA, conforman en sí una gran área de estudio donde se aplican también técnicas de otras áreas de la IA.

### *1.2.2 Robótica*

El nombre robot procede del término checo **robota** (trabajador, siervo) con el cual el escritor Karel Capek designó, primero en su novela y tres años más tarde en su obra teatral **RUR** (Los robots universales de Rossum, 1920) a los androides producidos en grandes cantidades y vendidos como mano de obra de bajo costo para liberar a la humanidad del trabajo. Hoy en día, el desarrollo de los robots se dirige hacia la consecución de máquinas que sepan interactuar con el medio en el cual desarrollan su actividad (reconocimientos de formas, toma de decisiones, etc.). Al oír la palabra robot, a menudo se produce en nuestra mente la imagen de una máquina con forma humana, con cabeza y extremidades. Esta asociación es fruto de la influencia de la televisión y del cine, cuyos anuncios o películas muestran máquinas con forma humana llamadas androides. Con el fin de mostrar como lucen realmente los robots en la actualidad, incluimos en el capítulo 2 una variada galería de imágenes.

La disciplina que se encarga del estudio y desarrollo de los robots es la **robótica**. Esta se centró, en primer lugar, en los robots de la llamada **primera generación**. Estos eran incapaces de detectar los estímulos procedentes del entorno y estaban limitados en las funciones a una secuencia predeterminada y fija, y fueron aplicados principalmente a la industria. Luego aparecieron los que constituyen la **segunda generación**, capaces de desarrollar algún tipo de actividad sensorial. Y más recientemente surgieron los prototipos multisensoriales que interactúan en un grado muy elevado con el entorno y que se agrupan en la **tercera generación**. En esta última generación, la robótica se sirve de disciplinas como la mecánica, la microelectrónica y la informática, además incorpora técnicas como el reconocimiento y análisis digital de las imágenes y el desarrollo de sistemas sensoriales.

El creciente desarrollo de los robots y su constante perfeccionamiento ha hecho que cada día se apliquen en mayor medida a los procesos industriales en sustitución de la mano de obra humana. Dicho proceso, que se inició hacia 1970, recibe el nombre de **robotización** y ha dado lugar a la construcción de plantas de montaje parcial o completamente robotizadas. Esto conlleva, según sus detractores, a la destrucción masiva de puestos de trabajo, mientras que para sus defensores supone la satisfacción de necesidades socioeconómicas de la población y lleva aparejado un aumento muy considerable de la productividad (no filosofaremos más sobre este tema en esta tesis).
El desarrollo de estos robots no es trivial, una tarea tan simple como la de quitar el polvo con una aspiradora y esquivar convenientemente los obstáculos que haya, no se programa tan fácilmente. El aspecto más importante para este caso es la detección de los obstáculos (que no siempre son los mismos ni están en el mismo sitio) y la maniobra para eludirlos y seguir trabajando con la aspiradora. En comparación, los robots industriales que realizan operaciones muy precisas y a veces complejas, no plantean tanta dificultad en su diseño y fabricación. Limpiar el polvo del suelo de un salón es más difícil que ajustar piezas en una cadena de montaje de automóviles. La diferencia se debe a lo estático del entorno en una fábrica con robots industriales.

Aunque podríamos escribir cientos de páginas acerca de la historia de los robots no queremos extendernos más de lo expuesto; hay mucha literatura al respecto que puede ser consultada por el lector interesado, y para la comprensión del contenido de esta tesis alcanza con esta breve reseña.

### *1.2.3 Arquitectura de un robot autónomo*

Un mal diseño mecánico del robot puede echar a perder todo el trabajo por lo que resulta importante tener en cuenta este aspecto, quizás sería bueno contar con un ingeniero mecánico en el equipo que pueda detectar fallas en el diseño rápidamente, evitándonos tener que esperar hasta tenerlo casi terminado para percatarnos del error.

El modelo básico de un robot inteligente y autónomo debe permitir:

1. medir propiedades del entorno a través de sensores;
2. realizar acciones físicas tales como manipulación de objetos o locomoción;
3. la presencia de una estructura de control que determine la acción a ser realizada en un momento dado.

Por eso un robot debe estar equipado con varios sensores para percibir su entorno y actuadores que permitan llevar a cabo acciones en el mundo. Tiene además una tarea particular para realizar, eso es lo que viene a representar la etiqueta "Descripción de la Tarea" en la figura 1.

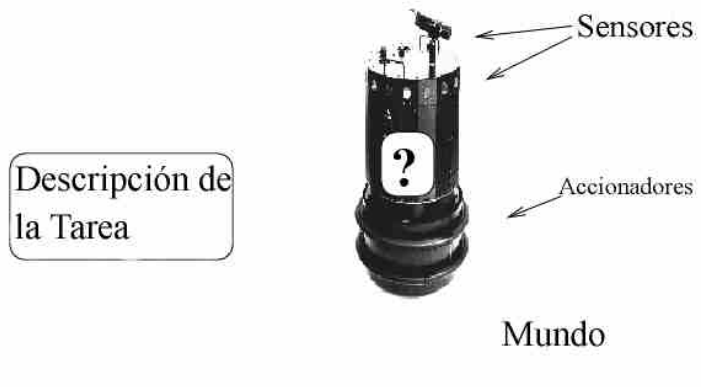



**Figura 1: el escenario del robot (extraída y adaptada de [Nehmzow and Mitchell 1995])**

A continuación nos ocuparemos de los sensores y accionadores. En la sección 1.2.4 veremos la *Descripción de la Tarea* y también lo que en la Figura 1 aparece como una interrogante.

### *Sensores (percepción)*

Los robots usan sensores para obtener información del mundo real. Varios de los sensores imitan nuestros sentidos, pero pueden también percibir cosas que nosotros no podemos, como campos magnético ú ondas ultrasónicas, planteando interesantes posibilidades para su desempeño.

En general, un sensor mide una característica del entorno y crea una señal eléctrica proporcional. Crear acciones de la vida real desde estas entradas limitadas es una de las tareas más difíciles para el diseñador del robot.

**Sensores para 'ver':**

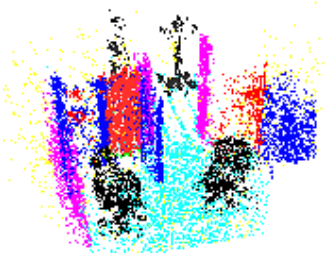
Los sensores de luz crean una señal proporcional a la luminosidad que captan, además se les puede agregar un filtro en el frente para una respuesta selectiva de manera que el robot "ve" únicamente un cierto color. También pueden usarse para navegación siguiendo una línea blanca (negra) sobre un piso oscuro (claro). Otros robots usan luz infrarroja para navegar (la misma que usan los controles remotos y es invisible); para ello emiten luz infrarroja, algunos rayos rebotan en un obstáculo y retornan al sensor de luz del robot indicando la existencia de un objeto. Cabe advertir que los sensores infrarrojos son sensibles al color de los objetos, esto significa que cuanto más oscuros son estos, más lejanos aparecerán para el robot.
Para sistemas de visión más complejos, los sensores de luz no son suficientes. Un robot que debe encontrar y remover productos con imperfecciones de una cinta transportadora necesita poder resolver rápidamente imágenes complejas y cambiantes. En esas situaciones, la imagen debe ser analizada por un programa de computadora para poder extraer, caracterizar e interpretar la información de las imágenes, para identificar o describir los objetos del entorno. La visión en la robótica es uno de los grandes desafíos para los ingenieros: es difícil programar a un robot para ver lo que es importante e ignorar lo que no lo es. Existen también problemas para interpretar cosas tales como resplandores, cambios de iluminación y sombras. Además para que un robot pueda tener percepción de profundidad necesita poseer visión estereoscópica como la nuestra y resolver dos imágenes ligeramente diferentes para construir una imagen 3D requiere una gran cantidad de memoria. Por si todo esto fuera poco, los sistemas actuales tienen una baja resolución, errores de paralaje, y la solución comúnmente usada de colocar la cámara encima del área de trabajo es bastante cuestionable para varias aplicaciones. Quizás si se montase el sensor de visión en la "mano" del robot tendríamos una mejor solución que también eliminaría esos problemas -siempre y cuando el robot tenga "manos"-. Hoy en día la imagen digital producida por un sensor de visión es simplemente un arreglo numérico que debe ser procesado para obtener alguna descripción explícita y con sentido de lo visualizado.

Algunos de los pasos del procesamiento de imágenes son:

- ***Preprocesamiento***: las técnicas de preprocesamiento lidian con la reducción de ruido y mejora de los detalles
- ***Segmentación***: los algoritmos de segmentación como detección de bordes, son usados para extraer los objetos de la escena
- ***Descripción***: los objetos son descriptos midiendo algunas características (preferiblemente invariables) de interés
- ***Reconocimiento***: los algoritmos de reconocimiento clasifican los objetos
- ***Interpretación***: los algoritmos de interpretación son usados luego para asignar un significado al conjunto de objetos reconocidos

A causa de lo complejo que resulta resolver el tema de la visión decidimos simplificar y no contemplarlo en nuestro robot. Es que siendo un área en sí misma, daría para hacer un proyecto completo abordando simplemente alguno de los múltiples aspectos mencionados anteriormente.

Esto es lo que vemos cuando entramos por una puerta....



…y esto es lo que el robot podría ver.
Este mapa 3-D fue producido en 1997 por un sistema de visión estereoscópico.

**Figura 2: Mapa 3D. Hans Moravec. Carnegie Mellon University**

**Primera clasificación de los sensores:**

- **Propio-receptor:** para medición de parámetros internos del robot (las "señales de vida" del robot)
- **Externoceptor:** para medición de parámetros externos al robot.

**Ejemplos de la primer categoría:**

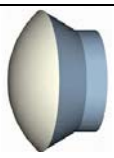
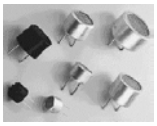
- **Posiciones de las articulaciones**: Inductosyn , Synchro, Resolvers, Encoders, RVDT (rotary variable differential transformer), Potenciómetros.
- **Carga de la batería**
- **Temperatura**
- **Atascamiento**: si el robot falla al detectar un obstáculo que le impide moverse sin este tipo de sensores el robot agotaría su batería inútilmente.

**Ejemplos de la segunda categoría:**

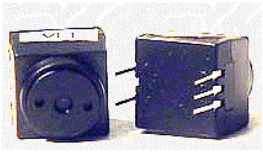
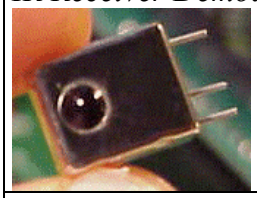
- **Contacto**:
  - Sensores de contacto
  - Sensores de fuerza/torque
  - Sensores de proximidad: ultrasónicos, ópticos (se basan en la modificación -por parte de objetos que están próximos- de la señal emitida)

- **Sensores de distancias a objetos:**
  - Time-of-flight: miden el tiempo que transcurre entre la emisión y el retorno de un pulso (los láseres y los sonares son los más conocidos de este tipo).
  - Triangulación: detectan un punto dado de la superficie de un objeto desde dos puntos de vista diferentes –dos puntos conocidos-. Dado que los dos focos desde los que se observa al objeto son conocidos, para obtener la distancia al objeto alcanza con una simple operación geométrica sobre los valores obtenidos.
  - Cámaras de video
- **Posición y orientación:**
  - Giroscopios: usan el principio de conservación del momento angular para mantener estable uno o varios ejes apuntando en la misma dirección exterior del giroscopio. Un giroscopio hace posible determinar cuán rápida es la rotación y cuánto ha rotado respecto a un sistema fijo de coordenadas.
  - Sensores de inclinación: son usados para determinar si el robot está a punto de volcarse.
  - Compases: pueden determinar la dirección del robot
  - Global Positioning Systems (GPS): son muy útiles para calcular la posición, basado en hasta 12 satélites.

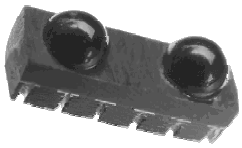
Concluyendo el tema de los sensores se enumera detalladamente un conjunto de sensores disponibles para todos los robots, aunque es claro que algunos no pueden ser usados en robots pequeños a causa del tamaño del sensor mismo.

| **Alcance (Range)** | | | |
|---|---|---|---|
| **Nombre** | **Descripción** | **Uso Típico** | **En la Robótica** |
| *Radar*  | Alcance[1] basado en el tiempo de vuelo de pulsos de radio. | Detección de obstáculos a través de la lluvia y nieve | La cantidad de cables y/o fibras ópticas dificultan la aplicación en robots pequeños |
| *Sick Laser*  | Basado en reflexión y triangulación | Cálculo de distancia a objetos | El aspecto limitante es el tamaño |
| *PiezoElectric Ultrasonics*  | Pequeñas piezas sensibles a vibraciones ultrasónicas, con un bajo consumo de energía, de gran sensibilidad, y baratas | Detectar ruido mecánico y vibración. | Su pequeño tamaño y bajo consumo las hacen ideales para robots pequeños. Se requiere electrónica para el filtrado de ruido y procesamiento de señales |

[1] El alcance es una función del tamaño de la antena (desde pulgadas a millas)

| **Creación de imágenes (Imaging)** | | | |
|---|---|---|---|
| **Nombre** | **Descripción** | **Uso Típico** | **En la Robótica** |
| *Cámara*  | Cámaras en miniatura, de bajo consumo (<30ma), ccd or cmos | Principalmente se usa para devolver imágenes remotas al operador de video | Tienden a tener baja resolución. Requieren procesamiento de imágenes onboard o trasmisión inalámbrica. La trasmisión inalámbrica agrega un nivel considerable de ruido. |
| *Cámara Digital* | Array nxn, array de fotoceldas, salida digital | Video web, toma de huellas digitales, video vía robot | Promisorio para robots pequeños porque las imágenes pueden procesarse localmente con poco esfuerzo, menor resolución que la mayoría de las cámaras |
| *Cámara IR* | Detecta luz por debajo del espectro visible, cercano al infrarrojo | Incendios, detección de pérdida de calor, detección de intrusos | Costosas, difíciles de hallar, requieren algo de electrónica para captura de video |
| *Omni Cámara*  | Cámara a la que se le agrega un espejo o lente para incrementar el fov (focus of vision) típicamente a 360º | Usado en robótica para proveer una visión de 360 grados | Poca resolución en la imagen, requiere un alto grado de calibración y el procesamiento de la imagen es más complicado |
| **Proximidad** | | | |
| **Nombre** | **Descripción** | **Uso Típico** | **En la Robótica** |
| *IR Receiver Demod*  | Receptor con amplificador y demodulador. Reducida sensibilidad al ambiente. | Controles remotos IR | Puede funcionar como detector de proximidad. Sensible a la superficie reflectante |
| *Capacitive Proximity* | Detecta la distancia[2] a los objetos basado en una combinación de los campos capacitivos. | Usado en máquinas industriales para detectores de posiciones relativas | Bueno para detección de colisiones o alineación de actuadores |

[2] El alcance está en el orden de los centímetros

| **Orientación** | | | |
|---|---|---|---|
| **Nombre** | **Descripción** | **Uso Típico** | **En la Robótica** |
| *GPS*  | Posición absoluta determinada por satélites | Originalmente para proveer posición absoluta en grandes vehículos. Reciente uso en aplicaciones embebidas | Todavía no se ha posicionado en la mayoría de las plataformas robóticas. No funciona bien en ambientes bajo techo. |
| **Comunicaciones** | | | |
| **Nombre** | **Descripción** | **Uso Típico** | **En la Robótica** |
| *IRDA*  | Permite la comunicación mediante rayos infrarrojos. | Usado para comunicación IR en 2 sentidos. Con modificación también detecta proximidad por IR | Todavía es sensible a la reflexión de la superficie |

## *Accionadores*

Los accionadores -también llamados actuadores o efectores- son las partes del robot que permiten modificar el estado del mismo o del entorno; son las "extremidades" y "músculos" del mismo.

La necesidad de estos dispositivos se debe a que un robot que simplemente esté sobre el escritorio todo el día no es muy interesante y no tendría diferencia con una máquina contestadora; los robots deben ser "criaturas" que se muevan en el entorno e interactúen con él. Los accionadores son típicamente elementos grandes y pesados que consumen mucha energía y pierden la mayor parte de la misma en la fricción. Mientras están funcionando, generalmente hacen ruido y/o se calientan, son las partes con mayor probabilidad de romperse. Puede resultar dificultoso trabajar con ellos al principio, pero son una parte esencial de cualquier robot.

El accionador más importante es el motor y le siguen en importancia las pinzas (grippers). El hecho de poder adaptarnos a la falla de alguno de los accionadores se considera una métrica de robustez (lo mismo ocurre para los sensores).

No es de extrañar que entre los accionadores se encuentren intentos por simular una mano humana. Existe una gran relación también entre sensores y accionadores puesto que los primeros además de su rol usual de percibir el entorno facilitan la operación de los últimos sobre la base del feedback que brindan sobre todo en las articulaciones, permitiendo controlar el movimiento y la fuerza que se aplica.

En definitiva, cualquier parte del robot que permita modificar el entorno (como por ejemplo unas pinzas), o el estado del robot mismo (por ejemplo las ruedas), es un

accionador. Cualquier herramienta que pueda ser agregada a un robot será considerada como tal.

**Clasificación de los accionadores llamados músculos:**
[*http://www.cis.plym.ac.uk/cis/InsectRobotics/Actuators.htm*]

- **Tradicionales**: hidráulicos y neumáticos. Pesan poco, son simples de adjuntar a un robot y tienen una gran potencia en distancias cortas. Tienen la desventaja de necesitar grandes compresores y baterías.
- **Reciprocating Chemical Muscle (RCM):** el músculo químico alternado es un mecanismo que aprovecha la densidad energética mayor de las reacciones químicas frente al almacenamiento de energía eléctrica. Es un dispositivo que convierte la energía química en movimiento a través de reacciones químicas no-combustibles. Tiene algunos beneficios: no requiere una fuente de encendido (permitiendo trabajar en ambientes explosivos), y es anaeróbico (permitiendo operar debajo del agua o en ambientes carentes de oxígeno).
- **Polímeros Electroactivos** (de siglas EAP en inglés): entre los materiales inteligentes activados eléctricamente, los polímeros electro-activos han sido el foco de mucha atención como la base potencial de los accionadores conocidos como músculos artificiales. Recientes testeos realizados por biólogos han confirmado que esos músculos artificiales pueden reproducir varias de las características de un músculo natural.
- **Aleación con memoria** (Shape Memory Alloys –SMA-): se denomina así a un grupo de materiales metálicos que han mostrado la capacidad de retornar a alguna forma (o tamaño) previamente definida cuando se los somete a un cambio particular de energía. En general, estos materiales pueden ser deformados a alguna temperatura relativamente baja y ante la exposición a una temperatura mayor, retornan a su forma anterior a la deformación. Los materiales que exhiben memoria sólo bajo calentamiento son referidos como teniendo memoria en un sentido. Algunos materiales también padecen un cambio en la forma cuando son sometidos a un re-enfriamiento. Estos materiales tienen una memoria en dos sentidos. Aunque se conocen una amplia variedad de aleaciones que exhiben el efecto de memoria de forma, solo aquellos que pueden recobrarse de grandes cantidades de tensión o que generan una fuerza significativa luego de haber cambiado de forma tienen un interés comercial. Hasta hoy las más usadas son las aleaciones de niquel-titanium y las basadas en cobre.

### *1.2.4 Cómo desarrollar aplicaciones para un robot*

Para entender cómo se encara la tarea de programar un robot hay que comprender los principios de la interacción robot-entorno que se muestran en la figura 3. Con este fin fue que explicamos en la sección anterior a los sensores y actuadores, pero dejamos la imagen del robot como una caja negra donde dada una entrada de los sensores se producía una salida en los actuadores. Aquí exploraremos qué ocurre en el medio entre los sensores y los actuadores, y cómo se enfrenta la tarea de elaborar una aplicación para un robot.

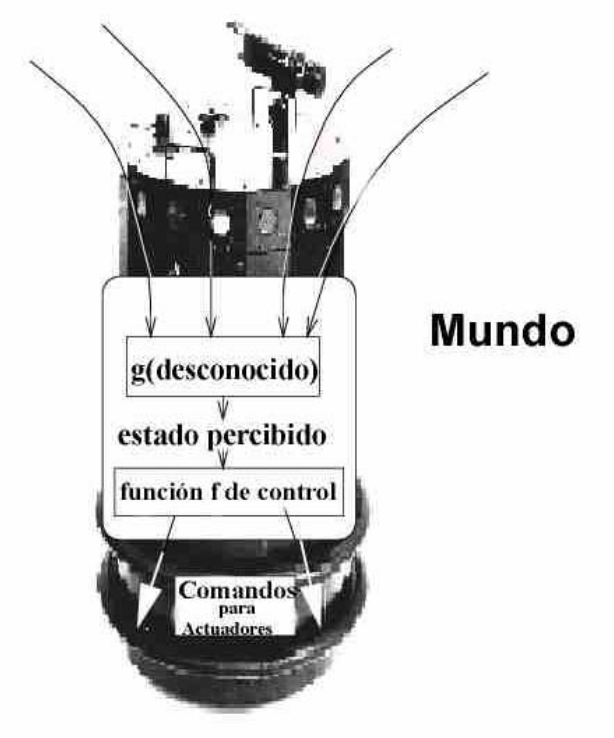



**Figura 3: Principios fundamentales de la interacción robot-mundo (extraída y adaptada de [Nehmzow and Mitchell 1995])**

Si se tiene en cuenta que en última instancia toda aplicación se reduce a leer de los sensores y enviar comandos sobre los actuadores se comprenderá que en principio es posible programar un robot con cualquier lenguaje de programación que nos permita leer los puertos donde estén conectados los sensores y escribir en donde están conectados los actuadores. Dado, además, que la mayoría de las universidades hacen sus propios robots, contamos con una gran diversidad de hardware. Es lógico entonces entender que no exista un lenguaje de programación especial para robots (aunque como se menciona en la sección 2.3.5 entre los proyectos del MIT hay uno que investiga el desarrollo de nuevos lenguajes). De hecho en la práctica se termina programando en bajo nivel (leyendo y escribiendo puertos, manejando el protocolo de mensajes específico del robot) o sobre una pequeña capa que permite operaciones como leerSensor(nro1) ó velocidadMotorDerecho(0.8).

Una primera consideración a tener en cuenta es que para que un robot realice una tarea no trivial, es necesario que esta se describa en términos de acciones primitivas sobre los actuadores del robot. Otra característica de este tipo de aplicaciones es que siendo el entorno tan variable, se hace impráctico desarrollar planes muy detallados de acciones antes de la ejecución, puesto que el mundo puede cambiar luego de que la ejecución comience e invalidar el plan. La forma más precisa (y comúnmente utilizada) para describir los objetivos del robot es definir los estados de los sensores que se deben alcanzar y los estados que se deben evitar (una descripción de los objetivos al nivel del hardware del robot). Como veremos en la sección 3.2, la definición de los estados tiene una influencia directa en la performance de los robots al momento de desenvolverse en el entorno. Es común medir el éxito de una aplicación en términos de metas específicas y no de la "elegancia de la solución"; es suficiente lograr alcanzar la meta razonablemente, aunque resulta difícil que se lo haga de manera elegante o de forma óptima.

La función *f* de control de la figura 3 es la que mapea el estado percibido del robot y del mundo en comandos sobre los actuadores: *f*: S→A, donde S es el estado actualmente percibido y A la acción elegida). Una primera aproximación a dicha función sería pre-programar una estrategia fija que mapee percepciones en acciones con ayuda de un programa de control estructurado del estilo clásico. Sin embargo al aumentar la complejidad de la tarea (o del entorno) aumenta la diferencia, en cuanto a la percepción del mundo, entre el robot y el diseñador humano, lo que dificulta determinar a priori cuáles señales de los sensores serán relevantes para una tarea particular y cuales no.

Aparece entonces el concepto de **aprendizaje**, donde se espera que el robot aprenda a actuar en el mundo y a desempeñar la tarea que le fue encomendada. El problema consiste ahora en diseñar un robot que mejore su desempeño a través de la experiencia (hay que definir en cada caso qué es *desempeño* y qué es *experiencia*). La forma más directa y simple es aprender la función *f* mediante ejemplos de entrenamiento correspondientes a pares entrada-salida de la función *f*. Esto resulta ideal para entrenar al robot mediante el uso de redes neuronales, pero hay varios casos donde no disponemos de ejemplos de la función *f*. Por ejemplo en el caso de un robot sin un entrenador humano que lo provea de ejemplos y que cuente únicamente con la habilidad de determinar cuando ha cumplido sus objetivos y qué recompensa está asociada con cumplirlos. En este caso no es posible aprender la función *f* directamente porque no tenemos disponibles pares de entrada-salida de los cuales aprender.

Una alternativa es aprender una función Q y usarla indirectamente para computar la función *f* de control; Q:SxA→V, donde V es el valor de efectuar la acción A en el estado S. En pocas palabras, Q es una función de evaluación sobre SxA. La función *f* puede computarse a partir de Q como la que maximiza el valor de efectuar la acción A en el estado S (maximiza la recompensa).

La naturaleza del entrenamiento necesario hace que se puedan diferenciar dos categorías de aprendizaje: supervisado y no-supervisado. La recompensa permite diferenciar entre aprendizaje con feedback inmediato y feedback retardado; en el primer caso la recompensa es provista en cada paso en la secuencia que lleva al objetivo, y en el segundo la recompensa es provista solamente al final de una larga secuencia de acciones que llevan a la meta. Evidentemente el aprendizaje con feedback retardado es más difícil y lento ya que no orienta al robot en la búsqueda de la solución, podría

encontrarla mejor si lo orientáramos al estilo de frío, tibio y caliente (eso es lo que hace el feedback inmediato).

Clasificación posible de clases generales de aprendizaje:

- **Aprendizaje por Refuerzo:** tiene sus raíces en los experimentos de Pavlov acerca de los reflejos condicionados. Pavlov mostró cómo con un estímulo real (que tenga una relación real) y un estímulo arbitrario (que no tenga una relación real) emparejado con algún comportamiento, podía evocar el mismo comportamiento ante la presencia del estímulo arbitrario y en ausencia del estímulo real (hacía sonar una campana cuando alimentaba a sus perros, luego de un tiempo comprobó que los perros salivaban al escuchar la campana aún en ausencia de la comida). Esto llevó a la idea de los conductistas de que una respuesta animal es una reacción o ajuste a estímulos o combinación de estímulos y respuestas. En la sección 1.2.4.1 trataremos el tema de aprendizaje por refuerzo con mayor detenimiento y es la clase de aprendizaje que usaremos al desarrollar una aplicación en el capítulo 3.
- **Métodos Evolutivos:** son particularmente interesantes para los interesados en el tema Vida Artificial. Se basan en la teoría de Darwin de la evolución de las especies. En la sección 1.2.4.2 comentaremos este tipo de métodos.
- **Aprendizaje Supervizado:** surge en los años 50 y 60 y representa el comienzo del uso de redes neuronales artificiales para el aprendizaje. Se entrena al sistema mediante la presentación de parejas entrada/salida; luego, ante la presencia de una entrada no vista durante el entrenamiento, se obtiene igual la respuesta que le hace corresponder el sistema. Durante el período de entrenamiento no importa demasiado la pobre performance del sistema ya que la etapa de evaluación es posterior al entrenamiento.
- **Métodos Individuales:** incluyen métodos de aprendizaje no-estándar, como por ejemplo aprendizaje por experimentación e imitación, entre otros.

A mediados de la década del 80 surgió una postura más radical referida a construir la inteligencia con un enfoque bottom-up: la *Subsumption Architecture*[3]. Esta arquitectura propuesta en [Brooks 1986a] tiene la visión de aproximarse a conductas de alto nivel (inteligentes) adoptando como modelo organismos vivientes de bajo nivel, como por ejemplo insectos, desarrollando así entidades (robots) que evolucionarán en su comportamiento emulando patrones de conducta, presentados por diversos organismos vivientes. Brooks considera que los investigadores de IA, quienes construyen todo su trabajo partiendo del hecho de que se cuenta con una representación simbólica del mundo, han fallado al momento de obtener resultados más destacables porque la parte más difícil de la tarea consiste en construir el mundo simbólico -que hasta el momento es realizado por un diseñador humano-, y debiera ser la tarea a la que se tendría que dedicar el mayor esfuerzo. Menciona, por ejemplo, que es más difícil para un robot interpretar una fotografía y abstraer los aspectos más importantes para la tarea que debe realizar, que actuar una vez que se ha hecho dicha abstracción. Realmente esa abstracción es la esencia de la inteligencia y la parte difícil de resolver del problema; pero hasta el momento lo resuelve el humano que diseña al robot, dejando poco por

[3] En algunos lados se ha traducido como Arquitectura de Subsumpción pero según la real academia española no existe esta palabra en el idioma castellano.

hacer al programa más que buscar la solución en un conjunto de soluciones posibles. Brooks piensa que un programa realmente inteligente debiera estudiar la fotografía, realizar la abstracción y resolver el problema. Por esa razón afirma que como realizamos las abstracciones por nuestros programas, éstos se desempeñan en un mundo de bloques en lugar del mundo real. Además, surge el problema de que nuestros sentidos ("sensores") no son iguales a los del robot, por lo que no percibimos al mundo de la misma manera que el robot que queremos programar. Esto lleva a que, en realidad, estemos intentando hacer una aplicación con **nuestra** representación interna del mundo y que, inclusive, nos olvidemos tal vez de algo importante en el proceso de describir introspectivamente nuestra representación interna y sea bastante diferente a la que usamos en realidad. Por esto, en este enfoque, Brooks se apoya el uso del mundo real como banco de pruebas para el robot, en lugar de usar entornos simulados por computadora (aun cuando haya usado también simuladores para comprobar sus hipótesis).

**Otras críticas al método clásico de resolver tareas:**

[Brooks 1991a] y [Brooks 1991b]

- Pueden haber cientos de bits de entrada y no hay un mapeo simple entre bits de entrada y el estado del robot y del mundo;
- Hay muchísimas posibles acciones que pueden ser tomadas en un momento dado;
- Al momento de elaborar la aplicación contamos con cierto conocimiento a priori sobre la tarea a realizar que no necesita ser aprendido;
- Los sensores presentan mucho ruido en sus lecturas, y no dan un mapeo simple desde el estado actual del mundo a un vector de entrada "limpio", que es lo que esperan los comportamientos desarrollados en simuladores
- Resulta imposible contar con un modelo interno que sea una representación completa del entorno y ello no es necesario para que el agente actúe adecuadamente

La *subsumption architecture* introduce el concepto de capas de comportamientos (ver la figura 4), Brooks y su equipo diseñaron hasta el momento los siguientes niveles:

0. Evitar el contacto con objetos (tanto si los objetos se mueven como si están estáticos)
1. Deambular alrededor sin golpear las cosas
2. "Explorar" el mundo percibiendo sitios a la distancia que parecen alcanzables y dirigirse hacia ellos
3. Construir un mapa del entorno y planear rutas de un sitio a otro
4. Percibir cambios en el entorno "estático"
5. Razonar sobre el mundo en términos de objetos identificables y realizar tareas relacionadas con ciertos objetos
6. Formular y ejecutar planes que involucran cambios del estado del mundo en alguna forma deseable
7. Razonar sobre los comportamientos de los objetos en el mundo y modificar los planes consecuentemente

La idea clave aquí es que las capas que controlan el sistema pueden correr en paralelo de forma asincrónica e interactuar mediante canales con poco ancho de banda cuando sea necesario. Cada capa individual corresponde a un nivel de habilidad; un nivel mayor de habilidad implica una clase de comportamiento más específico. Las capas superiores pueden suprimir o modificar las entradas de las capas inferiores y suprimir sus salidas también. Aunque el método de control distribuido permite que varios comportamientos corran en paralelo, también disminuye la posibilidad de contar con un control central para realizar tareas complejas. No hay una forma simple de tener un repositorio central; no hay variables que necesiten ser instanciadas; no hay reglas que deban ser seleccionadas y no hay elecciones que realizar. Tampoco hay un concepto práctico de aprendizaje aplicado a esta arquitectura. Simplemente reflejos e instintos.

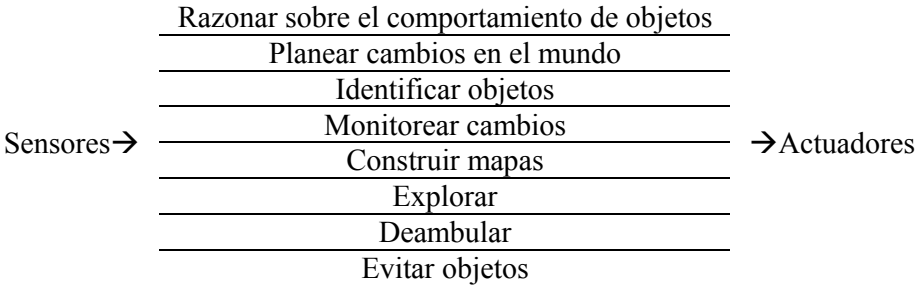



**Figura 4: una descomposición del sistema de control de un robot móvil basado en tareas**

Esta idea se sustenta en que la evolución biológica en la Tierra ha tomado más tiempo en desarrollar un comportamiento reactivo que suponga la supervivencia, la habilidad de moverse alrededor en un ambiente dinámico y las actividades propias para mantenerse con vida y reproducirse; que resolviendo comportamientos, lenguaje, conocimiento experto y razonamiento. Por esto, Brooks supone que una vez resuelto lo anterior, debiera ser simple resolver el tema del lenguaje, razonamiento, etcétera. En pocas palabras: que primero pueda sobrevivir, y luego razonar.

Como se comenta en [Brooks 1990], hay también problemas que son específicos de esta arquitectura: ¿Cómo combinar más de una docena de comportamientos generando módulos que sean productivos y cooperativos?, el tema de cómo escalar; ¿Cómo automatizar la construcción de interfaces para la interacción entre módulos generadores de comportamiento que permitan construir un sistema mayor?.

En la práctica, Brooks y su equipo han construido un robot físico con un máximo de tres capas y, en simulaciones, se ha llegado hasta seis capas corriendo en paralelo. Lo anterior muestra que esta arquitectura no resulta tan simple de construir ya que, teniendo aproximadamente 20 años, no ha producido resultados contundentes. Está igual que las ideas de cuya crítica surgió, que tenían unos 30 años en ese entonces y habían resultado infructuosas también.

Para averiguar si había algún avance -ya que no hallamos ningún paper de su autoría posterior a 1991 donde mencionara alguna mejora a su arquitectura- intentamos comunicarnos con Brooks, sin recibir contestación alguna a nuestro e-mail.

La aproximación de Brook (subsumption architecture) y las anteriores (con mundos de bloques) pueden ser complementarias. Pero como veremos en el capítulo 2, ninguna ha producido todavía resultados claros que nos permitan contar con robots dotados de un nivel de inteligencia siquiera comparable al de un insecto en todos sus aspectos (supervivencia y desempeño). Sí hay aproximaciones a ciertos comportamientos, pero no una emulación total. Nosotros usaremos el enfoque de mundo de bloques porque permite investigar el aprendizaje en los robots.

### 1.2.4.1 Aprendizaje por refuerzo (Reinforcement Learning)

Dentro de las categorías de aprendizaje, la más exitosa por sus resultados ha sido la de Aprendizaje por Refuerzo. Consiste, en pocas palabras, en el problema de aprender un comportamiento por ensayo y error. Este concepto se aplica al área de *machine learning* (un área mayor que la robótica), pero aquí nos limitaremos a lo aplicable para nuestro propósito: *robot learning*. Esta técnica presenta la promesa de ser una forma de programar agentes mediante premios y castigos según *qué* queremos que el robot haga, sin necesidad de especificar el *cómo* debe realizarse una tarea. Pero, si queremos cumplir en un 100% con el objetivo de programar sin especificar el *cómo* debe realizarse una tarea, debemos notar que en la práctica hay ciertas limitaciones. Por ejemplo, la cantidad de cómputo necesario para que el robot –comenzando sin conocimiento alguno- aprenda todo por sí mismo es formidable. Otro asunto a tener en cuenta es que la mayoría de los robots son sistemas de tiempo real por lo que las decisiones deben tomarse dentro de un tiempo crítico limitado o dentro de ciertas restricciones prácticas por lo que el proceso de elección de la mejor acción ante una situación determinada debe ser ágil.

En cada etapa de la interacción con su entorno, el agente recibe una entrada y alguna indicación del estado del entorno. Luego elige una acción para generar la salida. La acción cambia el estado del entorno y el valor de esta transición es comunicado al agente a través de una señal de refuerzo. El agente debe elegir acciones que tiendan a incrementar la suma de los valores de la señal de refuerzo. Esto lo puede aprender a hacer a lo largo del tiempo, mediante ensayo y error guiado por algún algoritmo. Es de esperar que, en general, el entorno sea no-determinista; es decir, que tomar la misma acción en el mismo estado, en dos ocasiones diferentes puede resultar en distintos estados siguientes y/o diferentes valores de refuerzo. Sin embargo, el entorno debe ser estacionario; es decir, que las probabilidades de hacer transiciones entre estados o de recibir señales de refuerzo específicas no cambien con el tiempo. Esta asunción es algo decepcionante ya que una de las motivaciones para construir sistemas que aprendan es el que operen en ambientes no-estacionarios. En la práctica la mayoría de los algoritmos son eficientes en entornos no-estacionarios que varíen lentamente, pero existe muy poco análisis teórico en esta área.

**Algunas características peculiares del aprendizaje por refuerzo:**
[Santos 1999]

- El problema tiene que poder modelarse como un proceso de decisión Markoviano,
- Cuando un agente elige una acción se le informa de la recompensa y el estado siguiente, pero no se le informa cuál acción debería haber elegido;

- El aprendizaje es concurrente con la evaluación del desempeño;
- Tiene la desventaja de que requiere que el espacio sea discretizable y almacenable en memoria; algo que resulta poco práctico la mayoría de las veces

Hemos comentado que los algoritmos deben ayudar al agente a actuar óptimamente, pero no mencionamos en qué debe ser óptimo. Por eso vamos a presentar distintos modelos.

## *Principales modelos de comportamiento óptimo*

[Kaelbling, Littman and Moore 1996]

- Modelo de *horizonte finito*, donde en un momento dado el agente debe optimizar su recompensa (*r*) esperada para los próximos *h* pasos:

$$E\left(\sum_{t=0}^{h} r_t\right)$$

  Este modelo no es siempre apropiado, ya que en muchos casos podemos no saber por adelantado el largo exacto de la vida del agente.

- En el modelo de *horizonte infinito disminuido* no se precisa conocer el largo de la vida del agente ya que tiene en cuenta la recompensa en infinitos pasos futuros, pero las recompensas que se recibirán en el futuro son geométricamente disminuidas de acuerdo a un factor $\gamma$ (donde $0 =< \gamma < 1$):

$$E\left(\sum_{t=0}^{\infty} \gamma^{t} r_t\right)$$

  Podemos interpretar el factor $\gamma$ de varias formas. Puede ser visto como una tasa de interés, una probabilidad de vivir otro paso, o como un truco para limitar la suma infinita. Viene a significar que una recompensa recibida k pasos en el futuro vale $\gamma^{k-1}$ veces, que es menos de lo que valdría si la recibiéramos ahora. Si $\gamma=0$, el agente es "miope" y solo le interesa maximizar la recompensa inmediata. A medida que $\gamma$ se aproxima a 1, más se toma en cuenta el futuro.

- Otro modelo posible es el de recompensa promedio, donde se supone que el agente toma acciones que optimizan a largo plazo su recompensa promedio:

$$\lim_{h\to\infty} E\left(\frac{1}{h}\sum_{t=0}^{h} r_t\right)$$

  Un problema con este criterio es que no hay forma de distinguir entre una política que obtiene una gran recompensa en las fases iniciales y otra que no.

## *Medición de la performance del aprendizaje*

[Kaelbling, Littman and Moore 1996]

Los criterios anteriores sirven para evaluar las políticas aprendidas pero también queremos evaluar la calidad del aprendizaje en sí mismo.

Hay varias mediciones (incompatibles entre sí) que pueden usarse:

- **Convergencia eventual al óptimo:** muchos algoritmos aseguran una convergencia asintótica al comportamiento óptimo. Esto es inútil en la práctica cuando tenemos un agente que alcanza rápidamente una meseta del 99% de optimalidad, el cual puede, en varias aplicaciones, ser preferible a uno que alcanza el óptimo sin duda, pero con una mayor lentitud en la tasa de aprendizaje.
- **Velocidad de convergencia cercana al óptimo:** esta medida precisa definir cuán cercano al óptimo es suficiente.
- **Arrepentimiento:** se mide el decrecimiento esperado en la recompensa obtenida al ejecutar el algoritmo de aprendizaje. Resulta difícil obtener resultados concernientes a estos algoritmos.

Lamentablemente, en la totalidad de las investigaciones que leímos, se mide el aprendizaje según el criterio subjetivo del humano que está diseñando el robot y observa el comportamiento aprendido; o se intenta medir la cantidad de recompensas/castigos luego del aprendizaje[4]; o se cuenta la cantidad de valores no nulos de la función de control aprendida. Es, sin duda, un área donde falta un gran desarrollo debido a la dificultad en el momento de definir cuál sería el comportamiento óptimo y de medir distancias al mismo. Podría ocurrir, incluso, que el robot hallara una solución mejor a la óptima que tenemos en mente, por lo que hay que elaborar métricas objetivas de optimalidad; independientes de un comportamiento particular.

## *Explotación vs Exploración*

[Kaelbling, Littman and Moore 1996]

Una de las mayores diferencias, entre aprendizaje por refuerzo (no supervisado) y aprendizaje supervisado, es que en el aprendizaje por refuerzo el agente debe, explícitamente, explorar su entorno buscando sus "ejemplos" de los cuales aprender. El problema que plantea explotación versus exploración está referido a descifrar, ante un estado determinado, cuándo debe explorar acciones no tomadas con anterioridad y cuándo explotar el conocimiento adquirido. El ejemplo clásico para explicar esto consiste en imaginar una sala de tragamonedas donde ya hemos jugado en algunas de las maquinitas; la cuestión es si nos quedamos con las que ya sabemos que nos dieron premio (explotar), ó probamos suerte con alguna de las que no intentamos todavía (explorar) con la esperanza de que puedan dar un premio mayor al que nos dan las que ya conocemos. En otras palabras, para obtener una gran recompensa, el agente debe preferir las acciones que ya intentó en el pasado y que le dieron resultado, pero para descubrir esas acciones debe seleccionar acciones que no haya intentado antes. Tiene que *explotar* lo que sabe para obtener recompensas, mientras *explora* para tomar mejores acciones en el futuro. Este problema se enfrenta con una técnica ad-hoc, donde se aplica alguna heurística para decidir cuándo explorar y cuándo explotar. Si el entorno es estacionario, es adecuado que estas técnicas converjan a algún régimen donde la exploración se efectúe raramente. Pero, si el entorno es no estacionario, la exploración debe de continuar para detectar los cambios en el mundo y adaptarse a ellos.

[4] Como mencionamos, esto no mide la performance de lo aprendido sino la política

## *El problema de la asignación temporal del crédito*

Otro problema que enfrenta el aprendizaje por refuerzo es, ante cada acción tomada, cómo saber si es acertada y si tendrá efectos a largo plazo (por lo que merece una recompensa), o no. Una posible estrategia es esperar hasta el final y recompensar a las acciones tomadas si el resultado fue bueno y castigarlas si el resultado fue malo. Esto presenta problemas en las tareas continuas, ya que es difícil saber cuál es el ”final”. Además, seguramente, no todas las acciones tuvieron que ver con el resultado final, por ello es difícil diferenciar cuáles tuvieron un efecto y cuáles no repercutieron. Otra estrategia es ajustar el valor estimado de un estado, basándonos en la recompensa inmediata y la estimación del valor del próximo estado (esta clase de algoritmos se conocen como *métodos de diferencia temporal* [Sutton 1988]).

## *Técnicas de aprendizaje por refuerzo*

[Sutton and Barto 1988] y [Robinson 2002]

Hay varias técnicas para resolver problemas usando aprendizaje por refuerzo.

- **Programación Dinámica:** Es una técnica para computar políticas óptimas. Por definición una política óptima puede ser utilizada para maximizar la recompensa. Las desventajas de la programación dinámica provienen de que requiere un modelo perfecto del entorno y esto puede requerir una considerable cantidad de computación. Incluso problemas simples pueden tener entornos con una cantidad de estados muy grande.

- **Monte Carlo:** Estas técnicas no requieren un conocimiento del entorno para nada. En su lugar están basadas en experiencia acumulada con el problema. Como indica el nombre, a menudo se usa el Método de Monte Carlo para resolver problemas que tienen una gran cantidad de elementos aleatorios. Una técnica posible sería mantener un registro de los premios recibidos después de cada estado y luego hacer que el valor de cada estado sea igual al promedio de toda la recompensa (positiva o negativa) encontrada siguiendo ese estado. Es posible resolver problemas usando Monte Carlo explorando cada posibilidad y luego generalizando una política óptima. Sin embargo, esto puede tomar demasiado tiempo.

- **Diferencia Temporal (Temporal Difference):** Uno de los problemas comunes de la programación dinámica y Monte Carlo (MC) es que las dos técnicas a menudo no producen información útil hasta que se visitan muchas veces un gran número de estados posibles. El método de Diferencias temporales (TD) fue formalizado en [Sutton 1988] pero tiene influencias de otros investigadores anteriores. Al igual que Monte Carlo, TD usa la experiencia para actualizar una estimación de la función de valor[5] a lo largo del tiempo. Como en MC, luego de visitar un estado ó un par estado-acción, se actualizará el valor de la función basado en lo ocurrido. Sin embargo, MC solo actualiza luego de que se ha completado la corrida del problema entero, o de algún episodio. Es en este punto

[5] Informalmente la función de valor da el valor de una situación dada; el valor de una situación es la recompensa esperada comenzando en esa situación siguiendo la política fijada.

donde MC vuelve y actualiza los valores promedios para todos los estados visitados, basado en la recompensa recibida al final del episodio. TD, en cambio, actualiza luego de que cada paso es tomado. El método general para TD básico, a veces denominado TD(0), es elegir una acción establecida según la política y luego actualizar el valor del estado actual. Dicha actualización se basa en la suma de la recompensa dada por el siguiente estado y la diferencia de valores entre el estado actual y el siguiente. Esta suma es, a menudo, multiplicada por una constante. TD funciona correctamente porque permite que el agente explore el entorno y modifique su función de valor mientras está trabajando en el problema actual. Hay un gran número de algoritmos de TD bien conocidos tales como Sarsa y Q-learning.

En la aplicación que desarrollamos y describiremos en la sección 3.2 implementamos un algoritmo que efectivamente usa *Q-learning*. Una de las ventajas de este algoritmo es que es insensible a la exploración, es decir, que los valores de la función Q convergirán a los valores óptimos, independientemente de cómo el agente se comporte mientras los datos son recolectados, siempre que los pares estado-acción sean examinados lo suficiente. Esto implica que, aunque el asunto de la exploración-explotación debe de considerarse en *Q-learning*, los detalles de la estrategia de exploración no afectarán la convergencia del algoritmo (afectarán la velocidad no la convergencia en sí). Estas razones hacen que sea el algoritmo más popular de aprendizaje por refuerzo. Igualmente, hay que tener en cuenta que espacios muy grandes de estados y/o acciones pueden enlentecer la convergencia a una buena política.

## *Generalización y Uso de Redes Neuronales (Neural Networks)*

La generalización es la aptitud para asociar una salida legítima a una entrada inesperada. En ocasiones nos encontramos frente a problemas con entradas y/o salidas continuas; ó frente a espacios de búsqueda tan grandes que el hecho de asumir que es posible enumerar todos los estados y acciones y almacenarlos en tablas para evaluarlos, los hace intratables, ya que -excepto en pequeños entornos- esto implica requerimientos importantes de memoria. Este problema, que se denomina el **problema estructural de la asignación del crédito,** se resuelve a través de técnicas de **generalización**, las cuales permiten compactar el almacenamiento de la información aprendida y transferir el conocimiento entre estados y acciones similares. La mayoría de estas técnicas usan redes neuronales. Se puede generalizar sobre las entradas (inputs), por ejemplo, a veces se particiona el entorno en regiones que se consideran iguales para el propósito de aprender y generalizar, pero sin conocimiento sobre el entorno es difícil realizar la partición apropiadamente. De ser necesario, se puede intentar generalizar sobre las acciones.

En [Touzet 1997], aparece un estudio comparativo de diferentes implementaciones de *Q-learning* usando un robot Khepera para el aprendizaje del problema de evasión de obstáculos. Touzet intenta desarrollar un método de *Q-learning* con una mejor generalización para el cálculo de los valores de la función Q asociados a un par estado-acción. Además implementa y compara *Weighted Hamming Distance*, *Statistical Clustering*, *Dyna-Q*, *QCON*, *Competitive Multilayer Perceptrons* y *Self-organizing Maps* con *Q-KOHON*. Son usadas dos medidas para analizar los resultados de los

experimentos: la distancia a los objetos a lo largo del tiempo y la proporción de movimientos ejecutados por el módulo de evasión de obstáculos que reciben refuerzos positivos. Los resultados sugieren que la performance de la generalización es mejor cuando *Q-learning* es implementado con técnicas neuronales tales como *Competitive Multilayer Perceptron* y *Q-KOHON*.

Cuando buscamos estimar los valores de la función Q usando una red neuronal es posible usar o bien una red distinta por cada acción (figura 5 de la izquierda), ó una única red con distintas salidas por cada acción (figura 5 de la derecha). Cuando el espacio de acciones es continuo, no existe estrategia posible. Otra posibilidad es usar una sola red con los estados y tener a las acciones como entradas y a los valores de la función Q como la salida. Entrenar dicha red no es difícil conceptualmente, pero sí lo es tratar con recompensas negativas y eliminar la asociación estado-acción que fue penalizada, sin interferir con el resto del conocimiento aprendido.

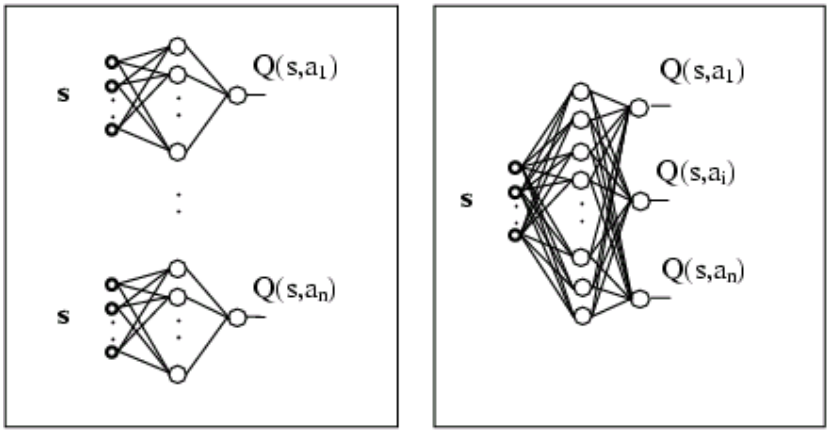



**Figura 5: dos posibles usos de redes neuronales para la implementación de la q-función [Santos 1999]**

El rasgo más importante de la implementación de generalización con redes neuronales es que se logra una buena generalización y requiere poco espacio para almacenar el conocimiento (a priori, solamente se necesita almacenar los valores de las conexiones entre neuronas, independientemente de la cantidad de pares estado-acción explorados).

## *Entornos parcialmente observables*

[Kaelbling, Littman and Moore 1996]

Para asegurar la convergencia, estos problemas de aprendizaje son modelados como Procesos Markovianos de Decisión[6] (Markov Decision Processes MDP). Cuando el entorno no es Markoviano, la probabilidad de la transición depende de la política que se está ejecutando, por lo que una nueva política conducirá a un nuevo conjunto de probabilidades de transición, por lo que no tenemos garantías de buenos resultados finales. Otro problema es que en muchos ambientes del mundo real no será posible tener una percepción completa y perfecta del estado del entorno para el agente. Por ejemplo,

[6] El modelo es Markoviano si los estados son claramente observables y la transición de estados es independiente de cualquier estado o acción anterior.

un robot podría tener que diferenciar si está en un corredor, en una habitación abierta, o ante una bifurcación T; y estas observaciones son propensas a errores. Este problema se denomina de "*percepción incompleta*" o "*estado oculto*". La solución más simple es no hacer nada en especial, es decir, tratar de hallar estados que se correspondan con estos casos e intentar aprender a comportarse, corriendo el riesgo de que no se encuentre nunca una solución, ya que los métodos no están garantizados para estos entornos. En general, este problema se resuelve agregando un componente aleatorio a las acciones del agente para evitar que se quede atascado en el pasillo por siempre. Hay otras soluciones propuestas, como por ejemplo, *Q-learning Recurrente* (usan redes neuronales recurrentes para aprender los valores de la función Q) que ha funcionado eficientemente en problemas simples, pero adolece de convergencia a óptimos locales en problemas más complejos. Hay otras propuestas, tales como *Sistemas Clasificadores, Estrategia de Ventana-histórica-finita* (permite decisiones basadas en la historia de observaciones de estados recientes y en algunos casos de las acciones también), *Estrategia POMDP* (incluyen el estado oculto en el modelo HMM –hidden markov model-).

## *Predisposiciones (biases)*

Las predisposiciones son incorporaciones del conocimiento a priori que tiene el diseñador de la solución.
Hay varios tipos de predisposiciones:

- **Modelando (shaping):** al robot se le presentan problemas simples para resolver y luego gradualmente se le plantean problemas más complejos
- **Imitación:** un agente aprende "mirando" lo que hace otro. En general, el otro agente es un humano y el robot aprende mediante los comandos que se le trasmiten desde un joystick
- **Descomposición del problema:** se descompone un gran problema en una colección de otros menores (divide & conquer)
- **Reflejos:** a veces lleva mucho tiempo al agente encontrar las partes interesantes del espacio para aprender. Estos problemas pueden superarse programando un conjunto de "reflejos" que hagan que el agente actúe inicialmente en una forma razonable
- La **función de refuerzo** guía al proceso de exploración, en este sentido puede considerarse como una predisposición también. La calidad de la función de refuerzo está intrínsecamente limitada por las cualidades del diseñador. Cuando un experimento de aprendizaje por refuerzo no converge, es imposible saber si se debe al hecho de que el experimento fue demasiado corto y se precisan más ejemplos, o si la naturaleza de la función de refuerzo impide la convergencia. Hoy en día los investigadores de aprendizaje por refuerzo utilizan un lento y doloroso encare por ensayo y error para definir la función de refuerzo

En general podemos clasificar las predisposiciones en tres categorías [Touzet 1998]:

1. Las que actúan en el nivel de búsqueda para mejorar la exploración;
2. Las que intentan reducir el tamaño del espacio de búsqueda;

3. Las que reducen el tamaño-base del aprendizaje simplificando el comportamiento que se busca.

Observación: evitar las predisposiciones que reducen el espacio de búsqueda tiene la ventaja de otorgar al robot un grado de flexibilidad tal, que permite al proceso de aprendizaje inventar soluciones útiles y novedosas que no sean las introducidas ni las esperadas por el diseñador. En general, la desventaja asociada con las predisposiciones es que limitan la expresividad del comportamiento que se obtiene como resultado. Al momento de incorporarlas hay que evaluar el costo (pérdida de expresividad) contra el beneficio (reducción de los tiempos de aprendizaje), pero, en definitiva, toda solución tiene en mayor o menor grado predisposiciones.

### 1.2.4.2 Programación Genética (Genetic Programming)

La programación genética proporciona un método que basado en la analogía con la evolución de las especies, genera **programas** de computación automáticamente. La idea central es similar a la de los algoritmos genéticos, pero en vez de manipular cadenas de "genes" se trabaja con los árboles sintácticos de los programas. Se genera un conjunto de programas-hipótesis y, mediante la mutación y recombinación de parte del conjunto al que se denomina "población actual", se obtiene la próxima generación remplazando una proporción de esta población por los sucesores de las hipótesis más "adecuadas". La adecuación se obtiene a través del uso de una función de evaluación que observa los resultados de su ejecución sobre los datos.

El objetivo de la programación genética es buscar, dentro de un espacio de hipótesis candidatas, la mejor de ellas. Donde la "mejor hipótesis" es aquella que optimiza una métrica predefinida para el problema dado.

El comportamiento básico es el siguiente: De forma iterativa se va actualizando la población de hipótesis. En cada iteración, todos los miembros de la población son procesados por la función de evaluación, tras lo cual una nueva población es generada. La nueva generación estará compuesta por:

- Las mejores hipótesis de la población actual seleccionadas probabilísticamente;
- Y el resto de las hipótesis necesarias para mantener la cantidad, que se consiguen mediante el cruce de individuos. A partir de dos hipótesis-padre seleccionadas probabilísticamente de la población actual, se generan dos hipótesis hijas recombinando sus partes siguiendo algún criterio establecido

Una vez llegados a este punto -con una nueva población, con el mismo número de individuos-, a un determinado porcentaje de la población se le aplica un operador de mutación.

Una de las dificultades que se presenta en la programación genética es el problema de muchedumbre (*crowding*). Se trata de un fenómeno por el cual las mejores hipótesis se reproducen rápidamente, de manera que una gran proporción de las nuevas generaciones se debe a éstas hipótesis y a otras muy similares (descendientes), reduciendo así la diversidad de la población y por lo tanto las posibilidades de la evolución. Una posible

solución -aunque no la única- consiste en usar una variación de la función de evaluación, denominada "ajuste compartido", de manera que el valor devuelto por esta función se devalúa ante la presencia de otras hipótesis similares en la población. Otra alternativa es restringir el tipo de hipótesis a las que se les permite la recombinación.

A diferencia de los algoritmos genéticos donde se evolucionan cadenas de largo fijo, la programación genética evoluciona programas de tamaño dinámico. Se basa en la premisa de que los programas de computadora pueden representarse como una estructura de árbol (mediante su árbol sintáctico).


**Figura 6: representación cómo árbol. La figura de la derecha es un ejemplo de un programa que computa (T1+T2)/T3 (extraída de [Dain 1998])**

En la figura 6 se muestra un ejemplo con funciones *f* y terminales *T*. Las funciones son operaciones que toman uno o más argumentos. Pueden ser secuenciales (i.e., +, *AND*), condicionales (i.e., *If, When*), o iterativas (i.e., *DoUntil, WhileDo*). Los terminales son operaciones que no tienen argumentos pero retornan un valor (i.e., variables y constantes).

Para aplicar programación genética a un dominio particular, es necesario que el usuario defina de antemano las funciones primitivas que se van a emplear, así como el tipo de los nodos terminales. En la mayoría de los casos, el rendimiento de la programación genética depende, básicamente, de la representación elegida y de la elección de la función de evaluación. Obviamente, se requiere que exista una solución para el problema en cuestión dentro del espacio de todos los programas que pueden ser creados usando el conjunto específico de funciones y terminales.

La **reproducción** consiste en copiar un individuo de la generación previa a la siguiente sin modificación alguna de su estructura (ver figura 7).


**Figura 7: la operación de reproducción [Dain 1998]**

La **mutación** se realiza seleccionando aleatoriamente un nodo de un individuo, removiendo ese nodo -con su correspondiente subárbol- y generando un nuevo subárbol al azar e injertándolo en la posición donde se removió el subárbol anterior (ver figura 8).

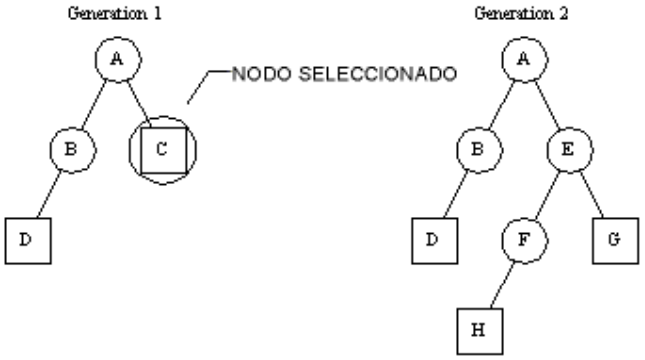



**Figura 8: la operación de mutación [Dain 1998]**

El **cruzamiento** involucra la selección dos individuos y la elección de un nodo al azar en cada uno de ellos. Los nodos seleccionados -con sus respectivos subárboles- son intercambiados entre los dos individuos (ver figura 9). Las probabilidades de reproducción y de cruzamiento son las que manejan el problema de la exploración-explotación.

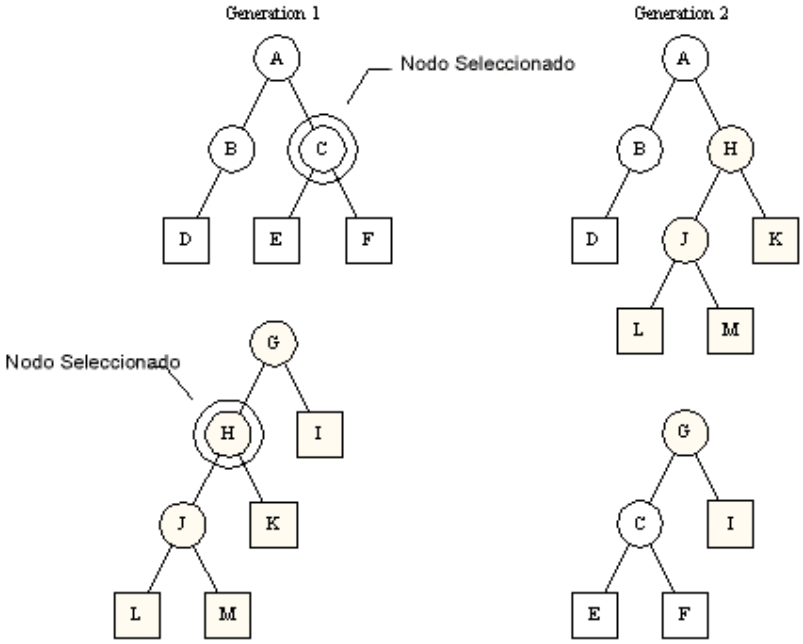



**Figura 9: operación de cruzamiento [Dain 1998]**

Hemos expuesto los conceptos principales de modo de hacer más comprensible la programación genética.

## *Notas*

La programación genética es un método de búsqueda en un espacio de soluciones. Supone un nuevo enfoque que permite abarcar todas aquellas áreas de aplicación donde se dificulte encontrar cómo resolver un problema, pero podamos estimar qué soluciones son buenas y cuales malas. A diferencia de los métodos de aprendizaje por refuerzo, aquí siempre se simula, ya sea por cuestiones de performance debido a que la evolución lleva mucho tiempo (estamos hablando fácilmente de días o semanas), ó por temor a que la mala performance inicial haga que el hardware del robot se dañe. Cuando se ha desarrollado toda la evolución hasta un grado suficiente, recién ahí se pasa a evaluar el programa sobre un robot real. Una desventaja que tiene es que no se adapta bien a los cambios en el entorno porque con la programación genética los robots no siguen aprendiendo una vez finalizada la etapa de evolución. Existen algunas investigaciones intentando solucionar este problema (ver la sección 2.3.1).

En ocasiones se mezclan los métodos de programación genética y aprendizaje por refuerzo, usando la parte genética para algunas cuestiones particulares, como desarrollar controladores de redes neuronales, ó para construir jerarquías de comportamientos desde comportamientos primitivos dentro de un algoritmo de aprendizaje por refuerzo. En definitiva, podemos llegar a tener una solución que use programación genética en el marco de un aprendizaje por refuerzo que, además, utiliza redes neuronales para generalizar. Lo importante es ver la mejor forma de combinar todo lo que conocemos utilizando las herramientas adecuadas allí donde funcionan mejor.

Han habido intentos de mezclar estas técnicas, por ejemplo en [Aljibury and Arroyo 1999] por medio de un algoritmo genético se intenta mejorar los parámetros[7] para crear la q-tabla, tarea que hoy en día es más artesanal que ingenieril dado que no existe una teoría que nos guíe. Pero, si bien consiguieron mejorar la solución inicialmente artesanal, lamentablemente, no lograron que los parámetros siquiera convergieran. Más bien, obtuvieron varios buenos resultados, pero con una gran variabilidad en los valores alcanzados.

[7] $\beta$ y $\gamma$ de q-learning, ver 3.2 por más información sobre q-learning

# 2. Estado del arte de la Robótica

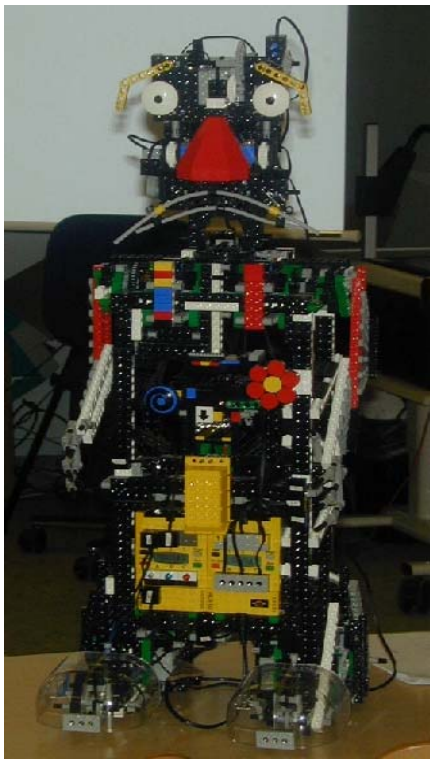
A continuación, mostraremos lo último en lo que a robótica se refiere. Lo enfocaremos desde tres puntos de vista: robots en las noticias; robots industriales; y robots en el ámbito académico. Intentamos obtener una visión actualizada del desarrollo de los robots. Buscamos responder a las preguntas ¿qué puede hacer un robot en la actualidad? ¿para qué son útiles? ¿qué se está desarrollando y en qué se está investigando?

## *2.1 Robots en las Noticias*

(información obtenida de diversas fuentes a través de Internet, tales como el sitio de la BBC de Londres [*http://news.bbc.co.uk*], el sitio de CNN [*http://cnn.com*] y [*http://cnnenespanol.com*], y búsquedas específicas de noticias sobre proyectos con Lego)

**Planta de Producción** [*http://www.daimi.au.dk/~baris/CaseStudy/*]
Se recurre a varios Lego RCX Brick para simular una planta de producción de acero belga. Es un caso de estudio en el proyecto VHS en Francia.

**Feelix** [*http://www.daimi.au.dk/~chili/feelix/feelix.html*]

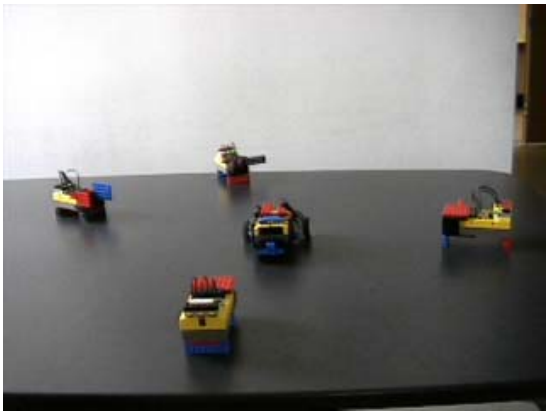


Hay un proyecto cuyo robot se denomina FEELIX, donde se estudia la posibilidad de exhibir varias expresiones emocionales en respuesta al contacto físico apuntando a generar una mejor comunicación con el robot. Se investiga en dos líneas: primero -admitiendo la importancia de la manipulación física- se indaga la interacción a través de estímulos táctiles; los variados tipos de estímulos que pueden despertar emociones en el robot se basan en un modelo de activación emocional apoyado en diferentes patrones de estímulo. Y segundo, los estados emocionales necesitan transmitirse claramente, por eso, implementaron expresiones faciales universales sobre un rostro caricaturizado (que construyeron con bloques Lego) y realizaron experimentos con niños y adultos. Se está trabajando para agregarle sonido que acompañe la expresión facial, y micrófonos para captar y responder a los ruidos, por ejemplo los aplausos. Están trabajando en conjunto con psicólogos de la Universidad de Paris XI.

**El Hermano menor** [*http://www.ceeo.tufts.edu/Me94/younger.html*]

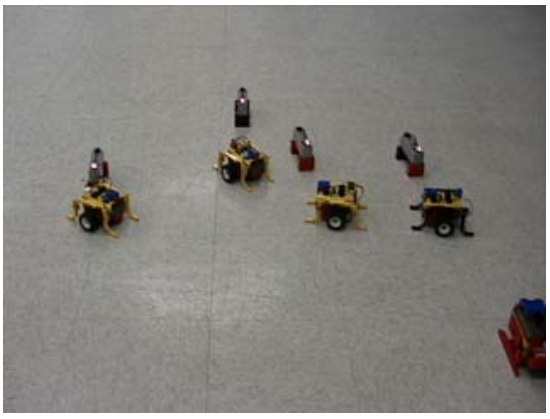
La meta de este desafío es manejar uno de los RCX mientras el otro hace lo mismo: se mueve o bien despacio o lento, hacia delante o hacia atrás, hacia la izquierda o la derecha. Esto se debe hacer asegurándose que ambos están dentro del alcance del emisor/receptor infrarrojo del RCX, ó bien haciendo que se comuniquen a través de Internet permitiendo que estén en lugares separados.

**Viajar con "Faros"** [*http://www.ceeo.tufts.edu/Me94/beacon.html*]



Se colocan cuatro RCX (son los faros) en una mesa. Cada RCX emite un mensaje simple ("espera 30 segundos", "gira a la derecha", ....). La nave nodriza (RCX #5) recibe el mensaje y obedece a cada faro siguiendo su orden y se mueve como le indicaron. Con este escenario es posible "programar" la nave según donde pongamos los diversos "faros".

**Cíclope** [*http://www.ceeo.tufts.edu/Me94/cyclops.html*]

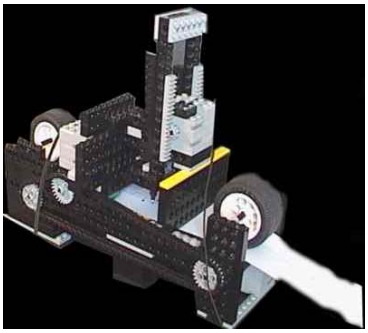


Poner cinco RCX en una tabla junto con obstáculos. Todos los RCX deben retornar el valor de la intensidad lumínica percibida a la computadora central. Esto se debe hacer mientras se mantiene el foco, es decir, que cuando un RCX se mueve detrás de un obstáculo debe pasar su mensaje a la computadora central a través de otro RCX. Si no está a la vista de otro RCX, debe deshacer sus pasos hasta que lo esté.

**RecojeLatas** [*http://www.geocities.com/EnchantedForest/Glade/1380/canbot.htm*]

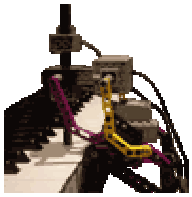
Este robot se mueve hacia delante hasta que una lata se desliza en el colector. Una vez que se haya deslizado y el sensor de luz dé una lectura por encima de un umbral determinado, se activa el micro-motor. Este levanta el brazo del colector hasta que la lata cae en el recipiente trasero. El recipiente puede almacenar dos latas de refresco. Un sensor de contacto es usado para avisar cuando el brazo del colector se levantó lo suficiente y el otro, que está en la parte baja del brazo del colector, es usado para avisar cuando el brazo bajó lo suficiente. Hay ciertos detalles técnicos necesarios para lograr la potencia suficiente para levantar la lata.

**Impresora** [*http://users.swing.be/philippe.jadin/mylego.htm*]



Esta impresora puede imprimir cualquiera de los 26 caracteres del alfabeto inglés. Usa papel estándar (utilizado por las máquinas comunes) y una lapicera. Además, utiliza 3 motores y ningún sensor. Tiene algún problema si la tensión del papel no se mantiene por encima de un mínimo "tirante".

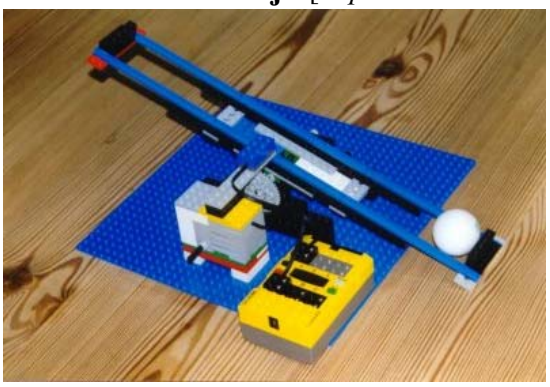
**Pianobot** [*http://www.jp3d.net/lego/pianobot.shtml*]



Un robot que toca el piano (el código está disponible)

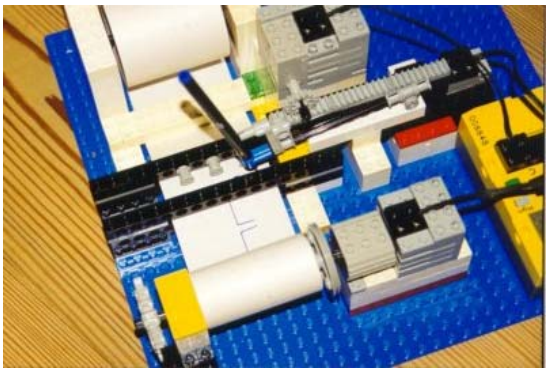
Los siguientes 6 proyectos son de Klaas Jan Huizing, un profesor alemán de lengua y se pueden encontrar en [*http://members1.chello.nl/~k.huizing/lego/index.htm*]. Es destacable lo que logró siendo un aficionado, estos proyectos demuestran la poca complejidad que hay detrás de estos artefactos. Estos se parecen más a robots industriales que a los del tipo autónomo que queremos usar en nuestra aplicación del capítulo 3. Se puede observar que se tratan todos de comportamientos reactivos.

**Bola en un subibaja** [*http://members1.chello.nl/~k.huizing/lego/seesaw.htm*]



(el código para el RCX está disponible). Esta máquina no tiene utilidad alguna, es solo para entretenimiento. Mantiene una bola moviéndose sin parar. Utiliza el riel del Lego y una pelotita de tenis de mesa. Y el subibaja evidentemente sube y baja continuamente. El sensor de luz detecta la bola para cambiar la dirección del subibaja.

**Servicio de Registradora** [*http://members1.chello.nl/~k.huizing/lego/device.htm*]



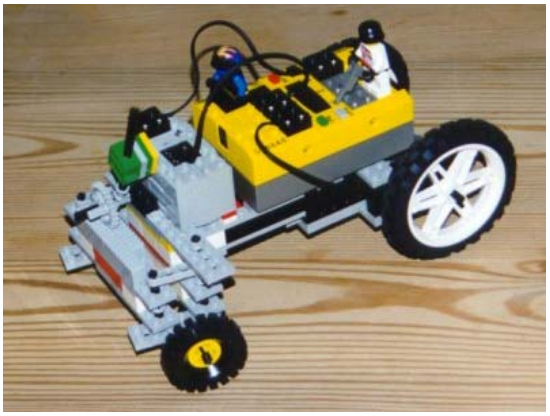
( el código para el RCX está disponible). Con un rollo de papel del tipo utilizado por máquinas registradoras y una lapicera se puede construir una máquina que dibuje en el papel cada diez minutos. Si ponemos el tiempo de inicio de la ejecución, podemos calcular el tiempo de un evento.

Por ejemplo, el investigador pegó el sensor de tacto a la puerta de la heladera. Cada vez que su novia o él abrían la puerta, la lapicera marcaba con diferentes símbolos en el papel. Luego de 24 horas, obtuvo una lista completa de la frecuencia y los momentos de uso del refrigerador. Puede usarse para contar cualquier cosa que se quiera: por ejemplo si usamos el sensor de luz podemos ver la frecuencia del uso de las luces.

**Dispensador de M&M** [*http://members1.chello.nl/~k.huizing/lego/dispen.htm*]

(el código para el RCX está disponible). Este dispensador usa las cajas pequeñas de M&M. Un brazo empuja la caja en la parte baja de la torre mientras se desliza. Un sensor de contacto hace que el motor cambie de sentido. El otro sensor de contacto detiene el motor cuando el brazo ha alcanzado la posición inicial. Cuando se comienza el programa, una cantidad aleatoria (entre una y tres) de cajas son expulsadas de la torre.

## Un auto que maneja en 8 [*http://members1.chello.nl/~k.huizing/lego/car.htm*]



(el código para el RCX está disponible). La dificultad está en la construcción y no en la programación. Un motor maneja el auto, el otro se encarga de la dirección. El sensor de contacto 1 es utilizado para avisar al RCX que las ruedas están alineadas.

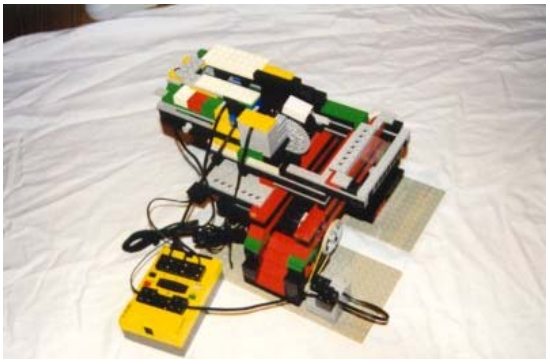
## Un auto que nunca se detiene

[*http://members1.chello.nl/~k.huizing/lego/never.htm*]

(el código para el RCX está disponible). El auto que maneja en forma de ocho le sirvió de base para este proyecto. Construyó una especie de paragolpes con tres sensores de contacto. Luego el motor, que maneja la detención del auto, gira la dirección. Maneja para atrás por 15 segundos (más o menos 180 grados de rotación), después, la dirección es enderezada nuevamente y el auto vuelve a manejar hacia delante hasta que toca algo.

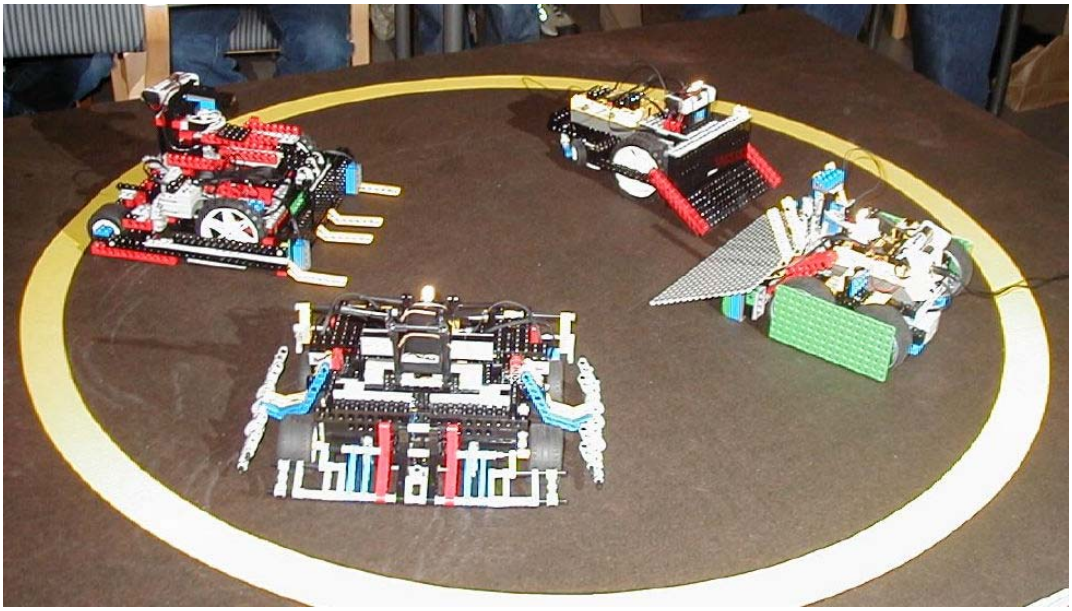
## Lego ordenador de ladrillos

[*http://members1.chello.nl/~k.huizing/lego/sorter.htm*]



(el código para el RCX está disponible). Usa dos cintas sin-fin a modo de cintas transportadoras. La cinta *A* transporta los ladrillos hacia el sensor de luz. El sensor distingue al ladrillo y detiene la cinta. Mide la luz y decide si es un ladrillo blanco o negro. La cinta *A* comienza a moverse de nuevo y deja caer al ladrillo en la cinta *B*. Si el ladrillo es blanco, la cinta se mueve hacia la izquierda. Si el ladrillo es negro, la cinta se mueve hacia la derecha. De esta manera, se forman dos pilas: una con ladrillos blancos y otra con negros. Éste es el último de los proyectos del muy ingenioso profesor alemán.

## Lego SUMO



Se trata de una competencia de Sumo. El ring es un anillo de un metro de diámetro pintado de amarillo sobre un piso negro. Los robots comienzan fuera del ring y una vez que entran, deben tirar a sus competidores fuera del anillo. Los rounds duran 3 minutos y en caso de empate, hay otro round de desempate. Para recompensar a los robots pequeños y ágiles, en caso de empate, el robot que haya rodeado al otro más veces se declara ganador. Hay algunas limitantes en cuanto a tamaño, peso y motores del robot.

En un apartado donde explican el desarrollo de un jugador, comentan que exploraron diversos modelos. Entre ellos una topadora y un robot pequeño con una lámpara en un mástil intentando engañar al contrincante para que salga por sí solo siguiendo la luz. Finalmente, optaron por uno que dieron en llamar *Dozer*. Le pusieron neumáticos con gran fricción para evitar que lo empujaran fácilmente hacia fuera y, dado que buscaban golpear al oponente de frente, probaron con varios sensores. Terminaron optando por un sensor solo y realizando barridos de 180 grados. Para detectar el ring y no salirse sin querer del mismo, utilizaron sendos sensores en cada rueda.

Hay muchas competencias entorno al Sumo entre robots (aunque están lejos todavía de las vistas en escenas de la película *Artificial Intelligence* de Steven Spielberg). Se puede hallar mucha información en Internet al respecto si resulta de interés, por ejemplo en:

- [*http://www.robotbooks.com/robot-sumo.htm*],
- [*http://www.robotroom.com/SumoRules.html*],
- [*http://www.roboworld.com.sg/prodcus.htm*].

## El lego Cube-Solver (que resuelve un cubo de Rubik de 3x3)

Usan dos RCX; se escanea cada superficie con la video cámara Vision Command; se calcula una solución en la PC (off-board) y se descarga la secuencia de movimientos sobre el RCX superior. Hay algunos trucos a tener en cuenta, como lubricar al *rubik* con una silicona en *spray* ya que su superficie es muy rígida para los legos; además hay que lograr el torque suficiente y las pinzas no agarran tan fuerte como debieran; también requiere precisión en los movimientos, reconocer los colores e ingresar el estado inicial del cubo (escribir este programa que reconoce los colores es la parte más larga del proyecto). En la página oficial de Mindstorm hay un hipervínculo donde se ofrece el código fuente: [*http://mindstorms.lego.com/eng/default.asp*]

## AIBO

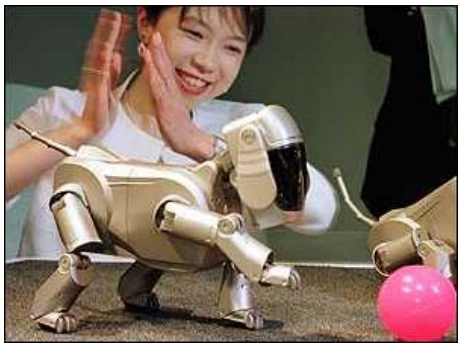


**Figura 10: Aibo**

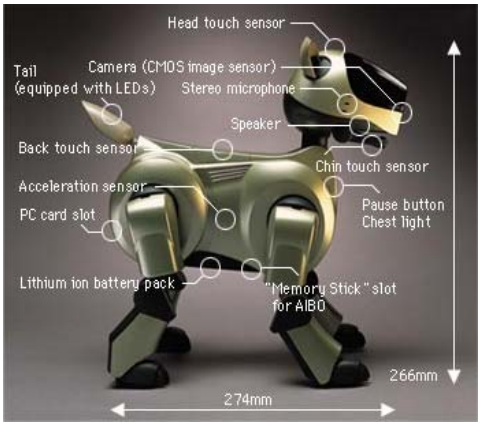



**Figura 11: descripción de sus partes**

Sony lanzó en 1999 a su perro robot Aibo (Aibo significa compañero en japonés). Han habido comentarios acerca de que la apariencia adorable del robot (ver figuras 10 y 11); por momentos pasa de ser una cosa a ser considerado un ser. Aibo presenta algunos aspectos destacables: se puede programar para que responda a un nombre; presenta una cámara en su interior (en su nariz) que se puede conectar de forma inalámbrica a la computadora permitiéndonos ver el mundo desde su perspectiva; cambiando su tarjeta de memoria se puede elegir un cachorro que lloriquea y demanda atención o el modelo de perro adulto. Existe un artículo cuyo periodista comenta que generó una respuesta emotiva en él, que sintió deseos de acariciarlo y tratarlo como a un perro real resultando claro que se corre el riesgo de confundir lo artificial con lo natural. Luego de un tiempo, el periodista se aburrió ya que la relación no variaba como lo haría con un perro real. Menciona que cabría la posibilidad de entrenarlo, pero reconoce que no le resulta interesante entrenar a un perro robot. Tuvo una decepción y problemas de relacionamiento ya que el perro robot no cumplió con todas sus expectativas. Concluye bromeando que como juguete es muy bueno pero que para ser una buena imitación de un perro le falta, ya que ni siquiera trae las pantuflas!.

Dado el éxito de ventas -según Sony vendieron más de 100.000 unidades hasta fines del 2001-, apareció en noviembre del 2001 una versión danzarina de Aibo con variantes como luces en su cabeza, las cuales se prenden cuando "escucha o ve algo que le gusta". Esta versión también reconoce más comandos verbales. Sony ha lanzado programas por separado que cambian el "temperamento" del robot en cuestión. Tiene una memoria de 32 Mbytes y una batería que le da autonomía por 90 minutos; cuenta con 20 motores; un procesador RISC de 32 bits; una cámara color a modo de ojos; un par de micrófonos estereofónicos por oídos; giroscopios para balance y un transmisor infrarrojo en el hocico para evitar golpearse contra la pared. Presenta algo que es ofrecido como emociones (6 diferentes): responde a los castigos y halagos; expresa su estado de ánimo visualmente con las luces de sus ojos y sonoramente con melodías y chirridos. El mejor truco que realiza es el de mirar, perseguir y patear una pelota rosada que viene con él en la caja. Hay que observar que para poder disfrutar de algún estado de ánimo (y no tener solamente un adorno) hay que comprar alguno de los cuatro módulos que se venden por separado. Por ejemplo, si compramos el módulo Life de Aibo, tenemos un cachorro recién nacido y a las 40 horas de atención aprende a caminar, responder y a realizar trucos. Acariciando su cabeza y diciendo "Buen muchacho" para reforzar el buen comportamiento ó palmeándolo cuando hace algo malo, vamos formando su "carácter" y "comportamiento". Con el tiempo se genera lo que llaman personalidad. Sony afirma que, luego de cierto período, cada dueño puede llegar a distinguir entre varios cuál es su perro por su "personalidad". El módulo Hello de Aibo ofrece, al contrario, un robot adulto sin la necesidad del tiempo de dedicación. Este permite que Aibo responda a un nombre a nuestra elección y entienda el comando verbal "Toma una fotografía". Podemos ver la foto -sacada desde sus ojos- insertando el cartucho de su memoria en una lectora conectada a una computadora con Windows como sistema operativo. Otros de los 40 comandos verbales que reconoce son "Sit", "Shake" y "Lay down" y al escuchar "Let's dance" se transforma en un "perro-disco" proveyendo su propia música y luces.

Pero Aibo no siempre responde. Según Sony estas muestras de actitud aumentan el realismo del robot, aunque en este caso, la "actitud" está apenas diferenciada del "no funcionar".

Existen muchos fanáticos en todo el mundo que han creado sitios web, magazines, clubes, herramientas de hacking para sacar lo que está guardado en la memoria, etcétera. Éste no se trata de un juguete para niños (de hecho Sony recomienda mantenerlo alejado de los pequeños); para algunos, es una alternativa que les permite tener una mascota sin la obligación de sacarla a pasear ni darle de comer. Esto hace pensar que quizás no haga falta esperar mucho tiempo para ver situaciones como la planteada por Isaac Asimov en su relato "El mejor amigo de un muchacho" donde se cuenta la relación de un muchacho con su perro robot mascota.

## NECORO

**Figura 12: NeCoRo**

Y si hay perros no podían faltar los gatos. Este se llama NeCoRo y es fabricado por Omron; tiene las siguientes desventajas: no puede caminar, no responde a comandos ni hace trucos. El creador se concentró, principalmente, en que brindara respuestas emocionales mediante sonidos y movimientos tales como ronroneos cuando es acariciado. "Si lo tienes mucho tiempo contigo, desarrolla una personalidad gentil. Pero si no juegas con él, te ignorará", agrega el encargado del proyecto. Se lanzaron unos pocos miles. Puede parar las orejas, entrecerrar los ojos, estirar sus piernas. Tiene sensores de tacto detrás y debajo de sus orejas y en su espalda. Además, posee una piel falsa que se contrae y expande con sus varios movimientos corporales y expresiones faciales (ver figura 12).

## Robot Pez

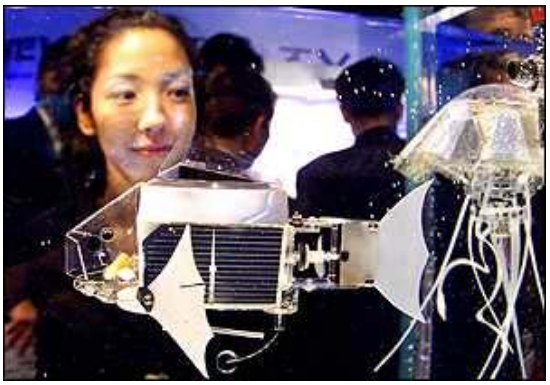


**Figura 13: robot pez**

La fiebre continúa con los animales: la empresa Takara presentó en el 2000 su serie de mascotas acuáticas Aquaroid que incluye un pez (figura 13), una medusa, un cangrejo y hasta una langosta.

## Robot insecto

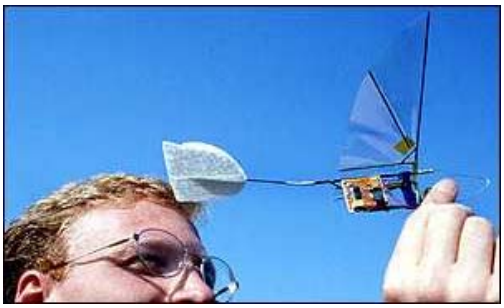


**Figura 14: robot insecto**

Los robots-insectos se apropian del aire (figura 14), pero esta vez no se trata de un juguete. La empresa californiana AeroVironment, la cual durante años ha construido micro-aviones espías, espera que en futuras generaciones puedan agregarle una mini-cámara a estos robots-insectos. Su "Black Widow", un avión de bolsillo de color negro y de aproximadamente 15 cm de diámetro, es usado por los militares de los

EE.UU para vigilancia detrás de las líneas enemigas. A cien metros del suelo es difícil detectarlo visual o sonoramente y puede enviar precisas fotografías del territorio que sobrevuela. Pero el último sueño del equipo es construir algo que pueda volar dentro de un edificio sin chocarse contra las paredes. Su avioncito actual vuela demasiado rápido para navegar con cuidado en un cuarto, por eso apuntan a construir un insecto robot. Las alas de los insectos, a menudo, se mueven en una compleja figura de 8 movimientos que da una excelente maniobrabilidad en el medio del aire pero que es difícil de imitar. Por esa razón apodaron a su último intento "MicroBat" ya que aletea más parecido a un murciélago que a un insecto. El mayor problema es construir unas alas que le proporcionen el empuje necesario para poder llevar un motor, circuitos eléctricos y una mini-cámara. Si bien afirman que todavía están lejos de lograrlo, no pierden las esperanzas. Quizás ya lo hayan logrado pero sea un secreto militar.

## Sony

**Figura 15: robot cantante y bailarin**

A mediados de marzo del 2002 Sony reveló su prototipo de un robot que puede cantar y bailar (figura 15). Además, puede quebrar sus caderas y mover sus brazos con ritmo. Aseveran que puede reconocer caras, nombres y voces; mantener conversaciones simples y levantarse cuando es tirado. Pero es un juguete caro que costará tanto como un auto de lujo. Afirman que una de sus características es su propia "personalidad". Comparte gran parte del software con otro producto de Sony: Aibo. Maneja un vocabulario de 60.000 palabras y puede realizar pedidos como por ejemplo: "Mantente quieto un minuto más mientras memorizo tu cara". También puede caminar sobre superficies con desniveles y acudir cuando es llamado. Está equipado con dos cámaras y puede diferenciar lo que es el borde de una mesa de lo que son marcas en el piso (algo que era difícil para Aibo con una sola cámara en su nariz). Insertando música y letra como datos en el robot, éste puede producir una voz cantante. La compañía agrega que el robot puede llevar a cabo bailes personalizados y complicados.

**Algunos datos técnicos**: tiene 60cm de alto y aproximadamente 6.35 kg de peso; cuenta con dos procesadores RISC y tiene capacidad para realizar movimientos de 38 grados de libertad. Además, posee cuatro sensores en cada pie y un giroscopio en el cuerpo que lo auxilian en sus caminatas. Puede reconocer las caras de 10 personas, e incluso, detectar emociones por sus gestos. Tiene siete micrófonos para escuchar cuando se le habla, pero aún presenta problemas si dos personas le hablan a la vez (efecto cocktail).

## Robonaut

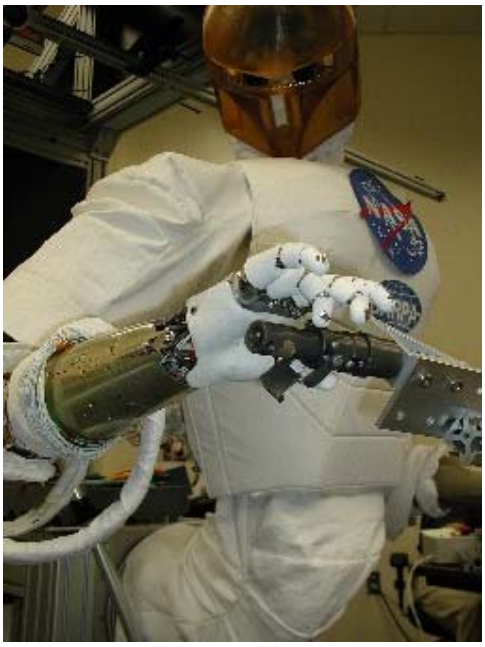


**Figura 16: humanoide de la NASA**

Robonaut (figura 16) es un humanoide diseñado por la NASA en colaboración con DARPA (Defense Advanced Research Projects Agency) . Necesita de la presencia de un operador humano que controle sus movimientos. Buscan que ayude en las tareas o realice aquellas que resulten muy riesgosas para nuestras vidas y en lo que se denomina Actividades Extra Vehiculares (EVA en inglés), comúnmente conocidas como caminatas. Tiene un especial esfuerzo de diseño en su habilidad y destreza manual, presenta una especie de sistema nervioso central con tecnología utilizada en la aeronáutica. Tiene sensores termales, de posición, táctiles, fuerza y torque (tiene unos 150 sensores por brazo). El sistema de control incluye una computadora a bordo. La guía la realiza a distancia un humano usando una estación de control de telepresencia. El robot tiene forma antropomórfica; está construido a escala humana; cuenta con dos brazos, dos manos con cinco dedos cada una, una cabeza y un torso. Trabajan con investigadores de diversas universidades norteamericanas. Están desarrollando la coordinación ojos-manos y un guante que permita detectar en qué parte de la mano ha habido contacto. También, a medida que aumenta la complejidad del robot, se hace difícil trasmitir la información de los cientos de sensores al operador humano, el cual se ve rápidamente abrumado por tanta información. Por esa razón se está trabajando para mejorar la presentación de la misma y eliminar la información que no resulte pertinente.

**ASIMO** [*http://world.honda.com/ASIMO/whats/index.html*]

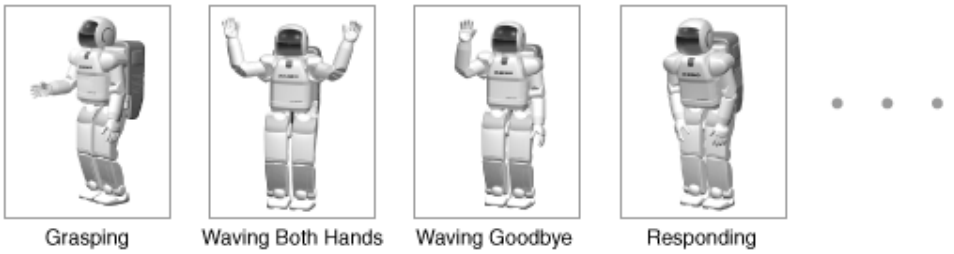



**Figura 17: ASIMO de Honda**

Honda, que no podía ser menos, también cuenta con su humanoide: ASIMO (*Advanced Step in Innovative MObillty*). Desde mediados de 1980, Honda comenzó a estudiar el desarrollo de humanoides, robots diseñados para lucir y moverse lo más parecido posible a un humano. Uno de sus primeros prototipos fue lanzado en 1996 y se denominó P2. Ya a fines del 2000, lanzó a ASIMO, este mide 1.20 mts, pesa 43 kg y puede caminar a una velocidad máxima de 1.6 km/h, tiene 6 sensores en cada pie, un giroscopio y un sensor de aceleración en el torso. Se trata de un robot japonés que todavía se está desarrollando y recién en el año 2002 pudo observarse por primera vez en EEUU. Fue hecho para que luzca lo más amigable e inofensivo posible (por eso también lo de su escaso tamaño), intentando superar la barrera de temor de los consumidores. No se trata de un juguete sino de un proyecto multimillonario. Puede expresar oraciones pre-programadas, reconocer y responder frente a 50 palabras en japonés (aunque todavía no puede mantener una conversación). También sabe realizar algunas tareas con sus manos como saludar o pasarle algún objeto a alguien (figura 17), pero su mayor logro es su caminar. Llevó 16 años de desarrollo para lograr la simulación de nuestro movimiento; tiene un centro de predicción de movimiento que mueve el centro de gravedad del robot antes del siguiente movimiento. Por el momento las compañías pueden alquilarle ASIMO a Honda para que salude en ciertos eventos (como ha hecho IBM), pero, evidentemente, Honda apunta a ofrecer en un futuro cercano un robot hogareño que haga los trabajos pesados o indeseados -más como mayordomo que como adorno-. Por el momento se sigue desarrollando.

## Telerobots

También hay un creciente interés en telerobots (robots dirigidos a distancia) para realizar tareas en centrales atómicas, actividades espaciales y otras, donde resulte menos peligroso y, en algunos casos más barato, poner un robot antes que a un ser humano. Además se están desarrollando robots industriales enfocándose, sobre todo, en la línea de ensamblaje, tendiendo a hacer desaparecer la necesidad de obreros en las fábricas (de los robots industriales nos preocuparemos más adelante en este capítulo).

## Mini robot [*www.sandia.gov/media/NewsRel/NR2001/minirobot.htm*]

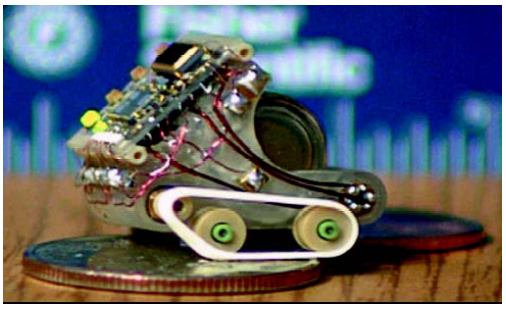


**Figura 18: mini robot**

Uno de los robots autónomos más chicos del mundo (ver figura 18) es, sin duda, el que elaboraron Doug Adkins y Ed Seller. Ocupa una superficie de 4 $cm^2$ y pesa menos de 28 gramos, tiene un procesador con una ROM de 8kbytes, un sensor de temperatura, tres pilas de reloj y dos motores para las ruedas. El cuello de botella en cuestiones de reducción del tamaño es en este momento la fuente de energía, ya que el cuerpo debe ser lo suficientemente grande como para contener las pilas necesarias para los requerimientos de energía. Es preciso que duren más y sean más pequeñas.

Las utilidades de estos robots serán varias, sobre todo cuando se logre miniaturizar y anexar otros tipos de sensores y cámaras. Todas ellas derivan de las actividades donde su pequeño tamaño es un factor a favor sobre sus robots colegas mucho mayores y voluminosos.

## Otras noticias

La experimentación en operaciones quirúrgicas con robots abre nuevos campos tan positivos como esperanzadores. La cirugía requiere de los médicos una habilidad, precisión y decisión muy cualificadas. La asistencia de artefactos puede complementar algunas de las condiciones que el trabajo exige. En operaciones delicadísimas, como las de cerebro, el robot puede aportar mayor fiabilidad. Últimamente, se ha logrado utilizar estas máquinas para realizar el cálculo de los ángulos de incisión de los instrumentos de corte y reconocimiento en operaciones cerebrales. Así mismo, su operatividad se extiende a la dirección y el manejo del trepanador quirúrgico para penetrar el cráneo y de la aguja de biopsia para tomar muestras del cerebro. Estos instrumentos se utilizan para obtener muestras de tejidos de posibles tumores que presentan un difícil acceso. Para ello resulta esencial la intervención del robot.

También hay noticias de intentos de mezclar seres vivos con partes robóticas: seres biónicos. El polémico y mediático científico inglés Kevin Warwick se realizó una cirugía para conectar su sistema nervioso a una computadora. Se puso una serie de electrodos cerca de su muñeca para intentar grabar y decodificar las señales que le permiten sentir y mover sus dedos. El experimento ha sido descartado por algunos expertos en neurociencia. Indican que el mismo tiene poco valor debido a que el tipo de señales que busca captar son muy débiles y están lejos de la sensibilidad de los aparatos con los que contamos en la actualidad. El profesor Kevin dijo que intentaría mapear las señales corresponden a movimientos de los dedos, sensaciones y emociones como shock, alegría e ira. Afirma que no podemos tomar este tipo de señales de animales puesto que éstos no pueden hablar ni comunicarnos qué sensaciones están sintiendo para que tomemos las medidas. Además probará el caso inverso, le serán suministradas a Kevin señales eléctricas para ver que tipo de movimiento generan en él. Prevé un

futuro donde las personas podrán ser mejoradas con una extremidad artificial o implantarse memoria extra.

## Otros tipos de robots

- los denominados task-oriented como el RL500 que corta el césped [*www.robomow.com*] o Carebot que aspira y limpia el piso [*www.geckosystems.com/carebot/rst.htm*].
- Otros ofrecen una telepresencia ambulante a lo largo de la casa o la oficina , ya sea a través de un mando remoto usando la handheld ó una interfaz remota. Un ejemplo es Amigobot [*www.amigobot.com/amigo/home.html*] que puede ser operado con una handheld ó a través de Internet por medio de una página segura que transmite imágenes visuales y auditivas de lo que el robot escucha y observa. Para mover el robot simplemente se apunta y cliquea, y para mover la cámara se usan las barras de desplazamiento.
- El Pathfinder que la NASA envió en 1997 a explorar Marte, consistió en una nave que, al llegar a Marte, se abrió como los pétalos de una flor. De ella salió un robot explorador que recogió información principalmente sobre la geología del planeta buscando indicios de vida.
- Se están construyendo grúas especiales para hacer que los sitios en construcción sean más seguros. Agregándole sensores, cámaras de video y un GPS (*global positioning system*) permiten ver los objetos y manipularlos con una mayor precisión que la actualmente manejada. Este proyecto es apoyado por la naval de los EE.UU (*US Navy*).
- Luego del derrumbe de las Torres Gemelas del World Trade Center de Nueva York en septiembre del 2001 se recurrió a robots para localizar sobrevivientes. Los robots fueron bajados por los huecos que había entre los escombros provistos de cámaras y sensores especializados para detectar el calor de cuerpos humanos o ropa colorida que se destacara entre el gris del polvo que cubría toda el área. Estos robots tenían la capacidad de poder pasar por sitios que serían muy peligrosos para los rescatistas o perros, tales como ventanales rotos ó hierros retorcidos. Se utilizaron una docena de robots de todo tipo: académicos, militares y hasta civiles acondicionados para ayudar en las tareas de rastrillaje. Igualmente se aclaró en su momento que, más allá de la tarea de ubicación, ninguno de estos robots podía tomar el lugar de un rescatista. No encontramos información acerca de lo efectivo que resultaron ser en la práctica, aunque por la falta de noticias informando lo contrario, suponemos que no sirvieron para ubicar sobreviviente alguno.

## *Notas*

Tenemos pocos robots domésticos hoy en día y se tratan todos de mascotas (perros, gatos y otros animales). El resto de las noticias nos informan de prototipos, éstos tienen principalmente fines militares y, en segundo lugar de popularidad, está el objetivo del entretenimiento.

Si bien estamos rodeados de promesas a corto plazo, pareciera que, por el momento, podemos contar solamente con robots-juguete y tendremos que esperar unos cuantos años para observar algo parecido a un robot-asistente. La gran mayoría son muy prometedores estéticamente (son lindos), pero técnicamente les falta complejidad. Es decir, que pueden parecer perros, gatos ó androides, pero distan mucho de tener la inteligencia necesaria para comportarse como tales. No encontramos incorrecto que se busque una estética. No está de más que se haga un esfuerzo para que parezcan cada vez más reales asemejándose así a lo que buscan imitar; de hecho, esto mejorará la aceptación de los robots entre los humanos. Pero para las investigaciones sobre algoritmos (sobre el "alma" del robot), no resulta fundamental una buena apariencia, razón por la cual no nos importa usar robots "feos" mientras nos brinden sensores y actuadores como para investigar.

Ha alcanzado con los ejemplos de los robots-perro, para entender tempranamente que desarrollaremos emociones hacia nuestros robots. Al establecer un vínculo afectivo, sumado a nuestro instinto de antropomorfizar todo, proyectaremos sueños y expectativas sobre ellos. Seguramente no faltarán los que luchen por los derechos de los robots y otras posibles reivindicaciones desarrollándose así toda una nueva cultura donde la barrera entre lo real y lo artificial se hará difusa. Si sumamos a esto la posibilidad científica de clonar seres humanos, este siglo nos deparará muchos debates filosóficos y humanistas sobre clones, robots, androides, hombres-máquinas, y máquinas-hombres. A medida que los robots se vuelvan más capaces de desarrollar ciertas tareas, habrá que redefinir qué es lo que nos hace ser seres humanos, de lo contrario, estos robots tendrán derecho a ampararse bajo nuestras leyes. Y habrá que meditar bien la respuesta ya que la capacidad de "pensar" puede dejar de ser una capacidad única de nuestra especie y lo mismo podría ocurrir posteriormente quizás con la capacidad de sentir.

## 2.2 Robots Industriales

Esta sección de robots industriales y, la siguiente, de robots académicos, son parte del material que utilizamos en una presentación que diéramos junto al Ingeniero Gonzalo Tejera en el Instituto de Computación de la Facultad de Ingeniería sobre robótica en el mes de octubre del 2002.

**Definición**

Mikell Groover, en su libro "*Automation, Production Systems and Computer Integrated Manufacturing"*, define al robot industrial como una máquina programable, de propósito general, que posee ciertas características antropomórficas, es decir, con características basadas en la figura humana.
Cabe destacar que la característica antropomórfica más común (diría que única) es en nuestros días la de un brazo mecánico, el cual realiza diversas tareas industriales. En la figura 19 podemos ver la analogía.

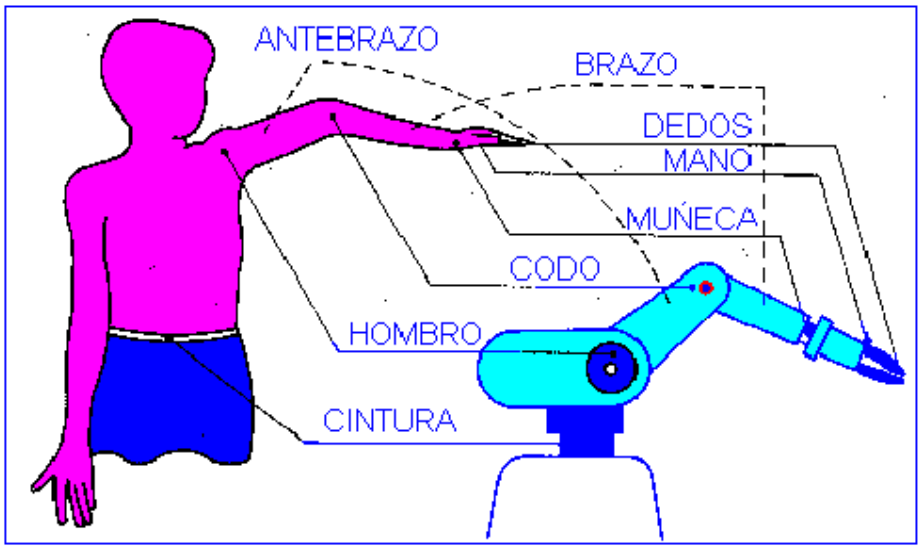



**Figura 19: brazo mecánico**

**Motivación**

Algunos objetivos de la robótica industrial son aumentar la productividad, optimizar el rendimiento de otras máquinas y herramientas relacionadas con la labor del robot, mejorar la calidad de los productos y disminuir los stocks de productos terminados así como sus plazos de entrega. Logran de esta forma beneficios importantes en las industrias y en sus procesos ya que los flujos en las cadenas de producción se han automatizado notablemente.

## Componentes

El componente principal es el brazo mecánico y consta de varias articulaciones y sus elementos (ver figura 20).

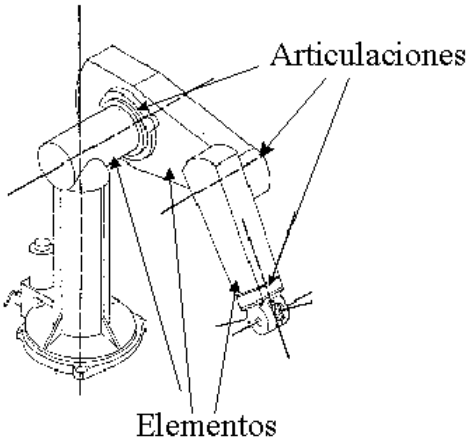



**Figura 20: componentes del brazo mecánico**

Las partes que conforman el manipulador y mostramos en la figura 21 reciben los nombres de: **cuerpo, brazo, muñeca y efector final.** Al efector final se le conoce comúnmente como sujetador o gripper.

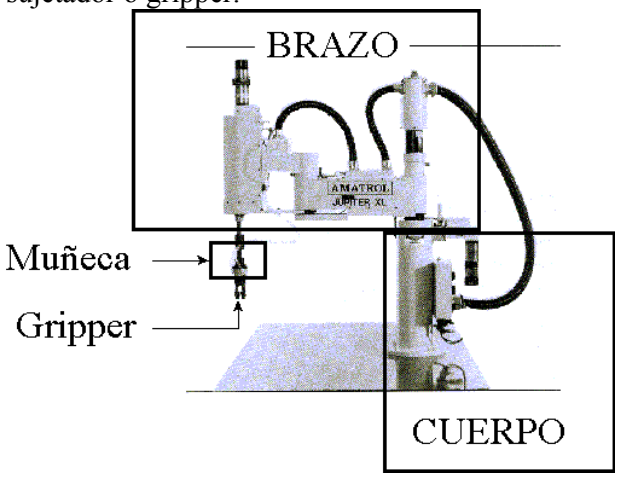



**Figura 21: partes de un robot industrial**

## Principales características de los robots industriales

- Grados de Libertad;
- Zonas de trabajo y dimensiones del manipulador;
- Capacidad de carga;
- Precisión de la repetibilidad;
- Velocidad;
- Coordenadas de movimientos;
- Tipo de actuadores;
- Programabilidad;
- Capacidad de memoria.

Vamos a centrar nuestra atención en las articulaciones. Cada una de ellas provee al robot de al menos un *"grado de libertad"*. En otras palabras, las articulaciones permiten al manipulador realizar movimientos:

- Lineales que pueden ser horizontales o verticales (figura 22).

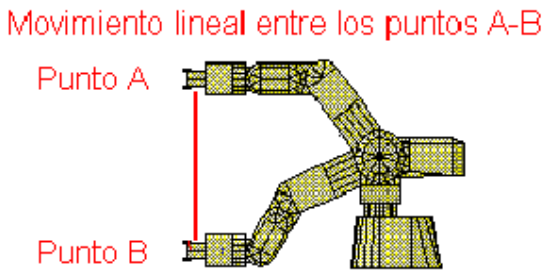



**Figura 22: movimiento lineal**

- Por articulación (figura 23).

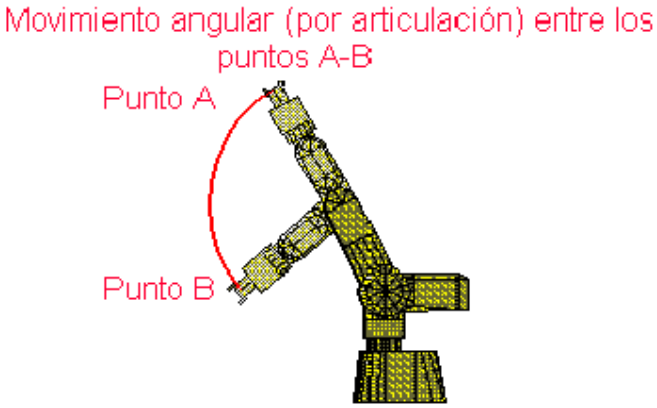



**Figura 23: movimiento por articulación**

(En los dos casos la línea roja representa la trayectoria seguida por el robot).

Nos encontramos con fabricantes de robots industriales en países como Hungría, Italia, Suecia, Luxemburgo, Bélgica, Reino Unido, Alemania, Suiza, EEUU y Canadá, entre otros. Y de empresas reconocidas tales como Bosch, Hitachi, Kawasaki, Mitsubishi, Panasonic, Peugeot, Seiko y Yamaha, entre otros.

## Variantes de robots industriales

Estos robots realizan tareas repetitivas y se emplean en gran escala en la industria automotriz, en la electrónica y en otras, donde se utilizan para armar o ensamblar automáticamente los respectivos productos, taladran, ponen componentes, los ajustan, sueldan, pintan, transportan piezas, etc.

Como ya mencionamos en general tienen la forma de un *brazo mecánico* donde se adapta en su extremo la herramienta que sea necesaria.

Los ***vehículos de control remoto*** pueden ser clasificados también dentro de la categoría de robots y se utilizan para movilizar herramientas o instrumentos en los sitios donde el hombre no puede acceder debido a las condiciones físicas o climáticas del lugar.

Los hay *terrestres, submarinos, aéreos y espaciales*, siendo estos últimos los más sofisticados. Podemos citar, como ejemplo, los robots que se emplean para construir túneles, apagar incendios, los que tienen fines militares, los mísiles teledirigidos, los vehículos espaciales teledirigidos o autónomos que permiten recorrer la superficie de un planeta o satélite, los que tienden cables submarinos, los que exploran el fondo del mar dirigidos desde un barco, etc.

Las ***prótesis para uso humano*** también pueden considerarse robots, ya que reemplazan funciones en los miembros inferiores y superiores de los seres humanos. Se han desarrollado verdaderas obras de arte en aparatos electromecánicos y electrónicos que realizan en forma parecida *el trabajo de las manos con sus dedos y las piernas*.

Los ***Manipuladores*** son sistemas mecánicos polifuncionales con un sistema de control simple y se emplean en tareas sencillas y repetitivas.

- Cuando el movimiento del robot es controlado directamente por el operador humano se dice que es un *Manipulador Manual*;
- Cuando el proceso -preparado previamente- se repite de forma invariable se dice que es un *Manipulador de Secuencia fija*;
- Y cuando se pueden alterar algunas de las características del ciclo de trabajo se dice que es de *Manipulador de Secuencia variable*.

Un ***robot de repetición o aprendizaje*** es un manipulador que repite una secuencia de movimientos que fueron *previamente ejecutados por un operador humano* haciendo uso de un dispositivo controlador manual. En la actualidad es el que más se utiliza en la industria y por el tipo de programación a que se hizo referencia, recibe el nombre de *gestual*.

## *Notas*

El hecho de que haya empresas grandes y conocidas vinculadas con el tema de la construcción de robots demuestra que la industria se está tomando muy en serio la fabricación de los mismos y que ya cuentan en la actualidad con los amplios canales de ventas que poseen dichas empresas. Por tratarse de última tecnología, es muy posible que todavía carezcan del número necesario de personas capacitadas suficientemente como para venderlos, desarrollar soluciones y realizar el mantenimiento post-venta. Igualmente consideramos muy limitado lo que puede hacer este tipo de robot ya que su programación es bastante pobre y carece de complejidad o sofisticación. Esto se debe a que, al no tener sensores, están pensados para desempeñarse en un ambiente controlado y seguro, sin mayores complicaciones ni sorpresas. Se puede pensar que están en una primera etapa de desarrollo, donde se comportan como los trabajadores de la película de Chaplin "Tiempos Modernos" ó como los de la Ford cuando Henry Ford incorporó la cinta sin-fin y se limitaban a realizar únicamente un tipo de movimiento repetitivo (y llegaban a ser los mejores aprieta tornillos del país!!). Aun así son revolucionarios para el momento actual; además que, por diversos motivos fáciles de deducir, para muchos dueños de fábricas es mejor que el trabajo lo haga un robot a una persona.

Aunque estos robots se están volviendo más comunes en los lugares de trabajo es importante recordar que no son "super trabajadores", sino herramientas que las personas usan y tienen varios inconvenientes reales:

- No son creativos o innovadores;
- No piensan de forma independiente;
- No toman decisiones complicadas;
- No aprenden de los errores;
- No se adaptan rápidamente a los cambios de su entorno.

Por esas razones se están investigando en el ámbito académico otros tipos de robots denominados "inteligentes". Su principal diferencia es que tienen la capacidad de relacionarse con el mundo real a través de sensores y de tomar decisiones adecuadas a las circunstancias (cambiantes) en tiempo real.

En la práctica los ***robots industriales*** son los que más aplicaciones útiles han tenido para la sociedad (más que los robots "inteligentes" que se encuentran en desarrollo) ya que los productos por ellos fabricados, por lo general, son para consumo masivo. Sin duda, esta tendencia se revertirá cuando se alcance la madurez requerida en las áreas de investigación necesarias para producir robots inteligentes más útiles.

## *2.3 Robots en el ámbito académico:*

En esta sección se enumeran varios proyectos de Europa y EE.UU. Con esto logramos obtener una idea aproximada de cuáles son las áreas que actualmente suscitan el mayor interés; también se alcanza a ver la madurez (o inmadurez en algunos casos) alcanzada en dichas áreas observando el nivel de dificultad de la tarea propuesta para ser resuelta.

También resulta interesante contrastar las diferencias en la organización de diferentes institutos del mundo. Algunos de éstos subsisten, casi exclusivamente, por el apoyo recibido por su colaboración con la industria; otros gracias al apoyo de organismos militares que verán alguna aplicación bélica para estas investigaciones y otros formando una red de colaboración entre Universidades.

A continuación se mencionan brevemente varios proyectos académicos divididos por países. No consiste en un estudio exhaustivo sobre lo que se está estudiando en cada país en sí, sino, más bien, en una muestra de algunos proyectos que elegimos por su originalidad buscando mostrar un abanico interesante de posibilidades de investigación.

### 2.3.1 En Suiza:

**EPFL (École Polytechnique Fedérale de Lausanne, Autonomous Systems Lab)**
[*http://asl.epfl.ch/home.php*]

***Autonomous Mini Robot "Alice 2002"***

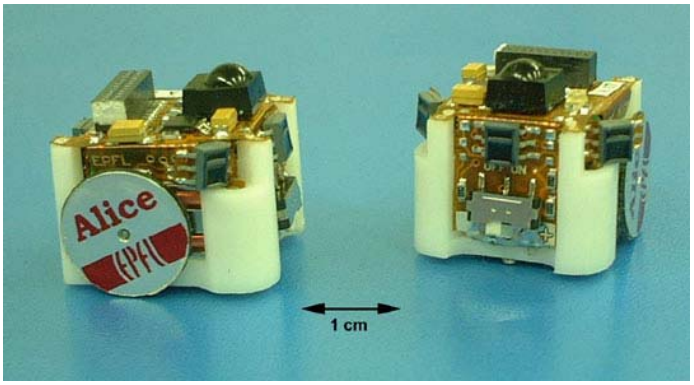



**Figura 24: robots alice**

**Motivación**

- Diseñar un robot inteligente lo más barato y pequeño posible;
- Estudiar el comportamiento colectivo con una gran cantidad de robots;
- Adquirir conocimiento en sistemas inteligentes altamente integrados;
- Proveer una plataforma en Hardware para futuras investigaciones.

**Descripción del Sistema**

- Microcontrolador PIC16LF877 con una memoria Flash programable de 8Kwords;
- 4 sensores IR de proximidad;
- Receptor de control remoto;
- Conector de 24 pin para extensiones, regulador de voltaje e interruptor de energía;
- Módulos de extensión:
  - Cámara de 102 píxels;
  - Comunicación bidireccional por radio;
  - Sensores de contacto;

**Proyectos y aplicaciones**

- Navegación y construcción de mapas;
- Equipo de fútbol: 2 equipos de 3 Alices jugaron al fútbol en una hoja A4;
- Sociedades mixtas robots-insectos.

**Ventajas**

- Tamaño pequeño (ver figura 24);
- Larga autonomía de energía (más de 10 horas);
- Flexibilidad frente al software.

**Especificaciones**

- Dimensiones: 22mm x 21mm x 20mm;
- Velocidad: 40mm/s;
- Consumo: 12 – 17 mW;
- Comunicación: local IR 6 cm, IR & Radio 10 mts.

## *Evasión de Obstáculos en un gentío*

**Motivación**

Los robots personales son una aplicación fascinante de las tecnologías de robots móviles. Las tareas posibles de los robots personales incluyen ayudar a personas minusválidas, apariciones en promociones como algo curioso, o guías turísticos. En vez de trabajar en un ambiente controlado (industrial), comparten el espacio con los humanos.

Antes de desarrollar robots personales en aplicaciones para el mundo real, ciertas cuestiones del movimiento de un robot y la interacción robot-humano deben mejorarse.

Esta investigación se concentra en los movimientos de los robots, específicamente en la evasión de obstáculos en ambientes dinámicos y altamente desordenados.

**¿Por qué un gentío?**

En primer lugar, un gentío es un buen ejemplo de un ambiente desordenado y dinámico. Segundo, es imposible evadir el factor humano, es un desafío para evitar obstáculos (las personas son "obstáculos" muy impredecibles). Muchos de los métodos de evasión actuales son poco confiables en estas condiciones.

**Objetivos**

Desarrollar un método nuevo para evitar obstáculos ante un movimiento rápido y refinado en ambientes dinámicos y desordenados. Será implementado y testeado con un robot guía-turístico en desarrollo en el laboratorio, el cual será desplegado en una muchedumbre.

**Puntos de partida**

- Comenzar con hardware de robot existente y software de bajo nivel;
- Existe un mapa del entorno (información estática geométrica global y topológica) ;
- Trabajos de localización.

## *Evolución de las reglas de Adaptación*

Desarrollaron un método para *evolucionar la habilidad de "aprender" on-line la manera de resolver una tarea, en lugar de evolucionar una solución para una tarea* como se hace normalmente en la computación evolutiva.

**Objetivos**

- Desarrollar sistemas evolutivos que puedan adaptarse on-line ante fuentes impredecibles de cambio;
- Explorar la evolución de nuevas modalidades adaptables para redes neuronales;
- Desarrollar la habilidad de aprendizaje incremental, i.e. de adquirir nuevo conocimiento y habilidades y almacenarlas.

## *Navegación Reactiva*

Primeramente propusieron este enfoque en 1996 y lo aplicaron a una tarea simple de navegación reactiva. El objetivo era desarrollar un controlador neuronal capaz de permitirle al robot avanzar tan recto como fuese posible mientras evitaba obstáculos. El robot fue puesto en el laberinto que se muestra en la figura 25.

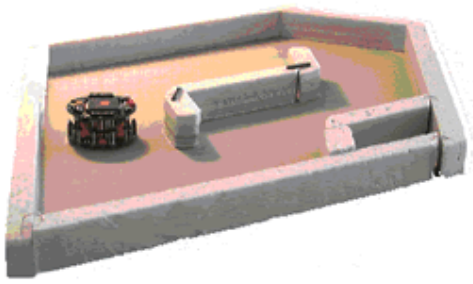


**Figura 25: entorno propuesto para navegación reactiva**

## *Coevolución Competitiva de Robots Adaptativos Depredador-Presa*

Considere el caso de un depredador y una presa (ambos robots) que coevolucionan en una competición. La aptitud del depredador está dada por la rapidez con la que puede atrapar a la presa, mientras que la de la presa está en función del tiempo transcurrido sin ser capturada por el depredador.

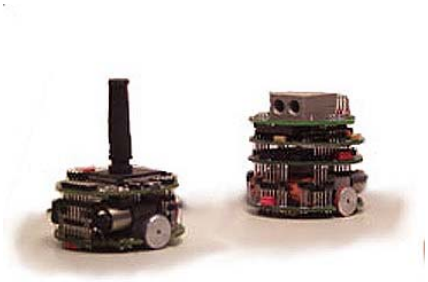


**Figura 26: depredador y presa**

El depredador (a la derecha en la figura 26) tiene sensores de proximidad alrededor del cuerpo y un sistema de visión, mientras que la presa tiene solamente sensores de proximidad pero puede moverse el doble de rápido. Si hacemos coevolucionar los pesos de los neurocontroladores de estos dos robots, observaremos cómo emergen interesantes persecuciones y estrategias de escape. Pero en todas las estrategias está todo absolutamente balanceado y después de varias generaciones el sistema coevolutivo entra en un ciclo donde, en algunas generaciones, gana el depredador y en otras gana la presa.

En un experimento realizado en 1997 quisieron probar que pasaba si hacían evolucionar los mecanismos evolutivos, en vez de los pesos fijos. También probaron experimentos donde el cromosoma podía codificar o bien reglas de aprendizaje, o cambios aleatorios en la sinápsis de la red neuronal utilizada.

**Los resultados principales fueron:**

- Al cabo de 20 generaciones, el depredador había adquirido la capacidad de buscar la presa y perseguirla, mientras que ésta escapaba moviéndose por todo el recinto. No obstante, como la presa era más rápida, esta estrategia no era muy favorable;
- Al cabo de otras 25 generaciones, el depredador localizaba la presa a distancia y acababa atacándola anticipándose a su trayectoria. A su vez, la presa empezaba a moverse tan deprisa a lo largo de las paredes que, el depredador fallaba a menudo y se estrellaba contra las paredes;
- Tras otras 25 generaciones el depredador había inventado la "*estrategia de la araña*". Se movía lentamente hasta la pared y esperaba a la presa, la cual se movía demasiado aprisa para detectar la presencia del depredador y evitarlo;
- Sin embargo, al dejar coevolucionar por más tiempo las dos especies de robots, constataron que redescubrían viejas estrategias que habían resultado ineficaces contra las que utilizaba su oponente en el mismo momento. Esto no es tan sorprendente ya que dada la simplicidad del entorno, el número de estrategias posibles para las dos especies de robots es limitado. Incluso en la naturaleza se observan casos donde ocurre esto, por ejemplo, en plantas e insectos. Se ha observado que al hacer el medio más complejo, por ejemplo, incorporando objetos al recinto, la diversidad de estrategias elaboradas por los robots era mucho mayor y se tardaba más en volver a las estrategias antiguas.

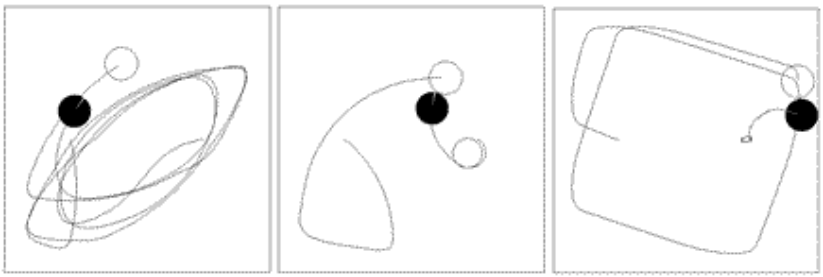


**Figura 27: tres evoluciones individuales**

En la figura 27 se muestran tres torneos de evoluciones individuales. El disco negro es el depredador, el blanco es la presa. Note el cuadro de la derecha. El depredador ha notado que la presa se mueve muy rápido alrededor de la habitación cerca de las paredes y, en lugar de intentar seguirlo, busca hasta que encuentra una pared y espera la rápida aproximación de la presa. Por los sensores de proximidad, la presa puede detectar al depredador sólo cuando está muy cerca (el depredador ofrece una superficie con poca reflexión), entonces, golpeará al depredador antes de poder alejarse. Esta estrategia es similar a la usada por las arañas para atrapar moscas.

## *Una tarea secuencial: El problema de prender la luz*

El robot Khepera equipado con un sistema de visión es puesto en una habitación con una lamparita (a la izquierda en la figura 28) y un interruptor de la lamparita (a la derecha en la figura 28). El interruptor está señalado con una raya negra pintada en la pared. La aptitud esta dada por la cantidad de tiempo pasado por el robot bajo la luz de

la lamparita cuando está prendida. Inicialmente la luz está apagada. Además, primeramente el robot debe ir hacia la banda negra para prender la luz. Las áreas negras y grises en el piso son usadas por la computadora para detectar, mediante los sensores posicionados bajo el robot, cuándo encender la luz y cuando acumular puntos de aptitud.

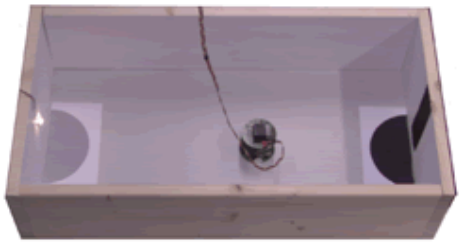


**Figura 28: entorno para el problema de prender la luz**

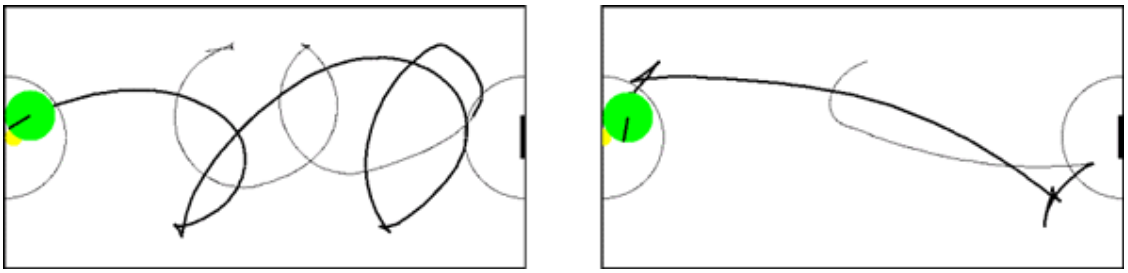


**Figura 29: muestras del desplazamiento**

En la figura 29 se pueden observar los comportamientos de un individuo genéticamente determinado (a la izquierda) y de un individuo adaptable (a la derecha). Note la diferencia de comportamiento. Incluso, el individuo genéticamente determinado, mostrado aquí, es bastante afortunado en su posición inicial y su dirección de rotación, una situación extraña.

## *Adaptación a Cambios Impredecibles*

Los investigadores aplicaron la programación genética a robots para que desarrollasen el comportamiento teniendo en cuenta las propiedades del entorno donde se encuentran. En consecuencia, tales robots resultaron ser capaces de adaptarse a una gran variedad de cambios impredecibles sin requerir evolución incremental. Testearon sus capacidades de adaptación para una variedad de nuevos entornos después de la evolución:

- **Transferencia de una Simulación a un Robot Físico**: evolucionaron sistemas de control en simulaciones y luego los transfirieron al robot real;
- **Cambiando el color de las paredes**: después de la evolución, cambiaron el color de la parte baja de las paredes de blanco a gris y luego a negro (recordar que como se comentó en el apartado de sensores, cuanto más oscura es la

superficie más lejana parece a los sensores infrarrojos del robot). El resultado fue que los individuos genéticamente determinados siempre golpeaban las paredes y se quedaban estancados ahí, mientras que los individuos adaptables lograban realizar toda la tarea desarrollando un comportamiento de evasión de paredes, el cual tomaba en consideración el color de las mismas.

- **Cambiando la disposición del entorno**: posicionaron el interruptor y la lamparita en posiciones diferentes a las vistas durante el entrenamiento evolutivo y hallaron que los individuos genéticamente determinados siempre fallaban al realizar la tarea. Esto se debía a que la trayectoria desarrollada estaba hecha a medida de la disposición del entorno en el que evolucionaron, como se observa en la figura 30 a la izquierda. En su lugar, los individuos adaptables lograron encontrar siempre el interruptor y luego avanzar hacia la lamparita, sin importar donde se encontraban, como se muestra en la figura 30 de la derecha.

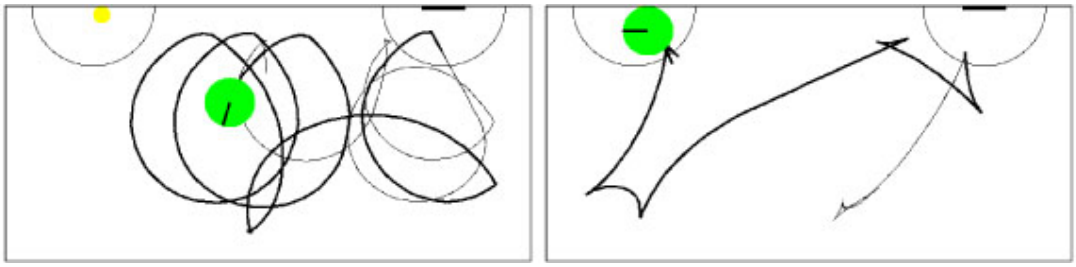


**Figura 30: muestras del desplazamiento**

## *Evolución de la Cooperación y División del Trabajo en las Hormigas Artificiales*

Las hormigas componen cerca del 15% de la biomasa animal en la mayoría de los entornos terrestres. Dos aspectos claves de su enorme éxito de supervivencia y proliferación son la habilidad de cooperar y de realizar una eficiente división del trabajo. Estos rasgos, que permiten a las hormigas realizar tareas complejas en diferentes entornos, serían de gran beneficio para los robots autónomos.
Es sabido que la jerarquía juega un rol importante en propiciar la evolución en insectos sociales, por eso, quisieran determinar si el rol de la jerarquía puede demostrarse experimentalmente con robots. Además, esto permitirá inferir algunas directrices para el diseño de agentes autónomos (robots) capaces de cooperar y auto-adjudicarse tareas.

### Métodos

Se investiga el asunto de la cooperación y división del trabajo haciendo "evolucionar" colonias de "hormigas artificiales" implementadas en robots móviles pequeños y por medio de simulaciones en software. En particular, están usando un simulador llamado Webots (se muestra una pantalla de este simulador en la figura 32), fue desarrollado por Cyberbotics. [*http://www.cyberbotics.com*]), para simular a los robots ALICE (figura 31) [*http://dmtwww.epfl.ch/isr/asl/projects/alice_pj.html*].

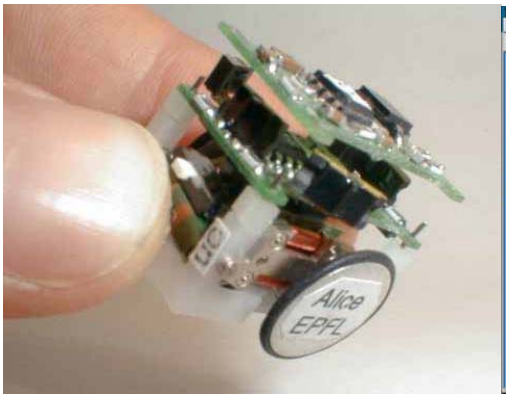



**Figura 31: robot Alice**

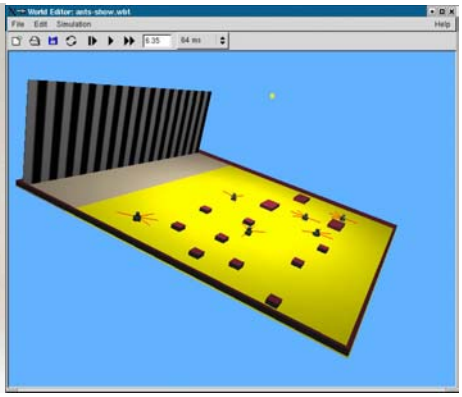


**Figura 32: vista del simulador Webots**

## Experimentos

El primer conjunto de experimentos que se ha propuesto está diseñado para determinar en qué medida el nivel de cooperación entre robots es influenciado por la composición del grupo (nivel de jerarquía). Para testear el rol de la jerarquía trabajan con cuatro tipos de colonias experimentales[8] desempeñando una tarea de búsqueda de comida, mientras varían la composición y nivel de selección (artificial). En la organización, los robots deben buscar comida y llevarla al nido (o nido). Hay dos tipos de comida: la pequeña, que puede ser arrastrada por un robot-hormiga simple y la comida grande, que requiere dos robots para ser transportada. Los robots tienen la posibilidad de pedir ayuda cada vez que hallan una comida del tipo grande.

En una segunda y tercera fase del proyecto, estudiarán bajo que condiciones emerge la división del trabajo y el papel del tamaño del grupo en la evolución de dicha división.

## Organización del proyecto

Este proyecto está organizado en cuatro partes principales:

- Diseño de organizaciones experimentales e implementación de algoritmos de evolución artificial (en progreso);
- Diseño e implementación de una nueva versión de los robots ALICE, adaptados para este proyecto (en progreso);
- Experimentos evolutivos usando el simulador Webots;
- Testeo de los resultados de la evolución simulada usando robots reales.

# *Robots Voladores adaptables basados en la visión*[9]

La visión robotizada plantea, como ya comentamos, el tema de cómo usar eficientemente -y en tiempo real- la gran cantidad de información obtenida de los

[8] No se mencionan particularmente las diferencias entre los cuatro tipos de colonias.
[9] Proyecto para el PhD de Jean Christophe Zufferey.

sensores. Para sistemas conductistas que deben reaccionar rápidamente ante su entorno, no es viable utilizar el principal enfoque, el cual consiste en computar la visión basada en una secuencia de pre-procesamientos, segmentación, extracción de objetos y reconocimiento de patrones en cada imagen.

En este proyecto exploran un enfoque por el cual emergen comportamientos robustos basados en la visión, mediante la coordinación de varios componentes que pueden vincular directamente aspectos visuales simples con comandos para el motor. La inspiración biológica es tomada de la visión de los insectos y son usados algoritmos evolutivos para desarrollar redes neuronales que desenvuelvan comportamientos eficientes.

**Figura 33: vistas del robot volador**

Construyeron un prototipo capaz de volar autónomamente en espacios interiores. Esto simplificó los experimentos ya que evitó el efecto del viento y la dependencia del clima y permitió modificar el entorno visual del robot según se iba requiriendo.

**Características de este robot:** peso total 47 gramos, 82 cm de largo de las alas, 1.4 m/s como velocidad mínima de vuelo y espacio mínimo para volar aproximadamente 8x8 metros. Podemos observar dos imágenes en la figura 33.

## *Robot inspector del conducto de aire*

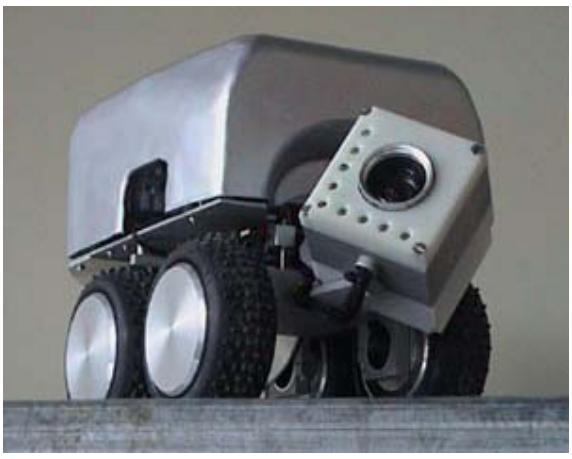


**Figura 34: vista del robot inspector de conductos de aire**

## Motivación

A pesar del gran número de robots disponible para inspección de tuberías, no hay en el mercado un sistema para la inspección de los ductos de aire acondicionado. Estos conductos tienen características muy diferentes a los ductos subterráneos: tienen muchas curvas, un fuerte flujo de aire, normalmente no hay agua en ellos y pueden tener secciones cuadradas o circulares.

## Descripción del sistema

- Cuatro ruedas con motores integrados y un codificador (figuras 34 y 35);
- Una cámara color (zoom 10x);
- Cámaras delante y detrás con iluminación;
- Cable para trasmisión de video/datos/potencia;
- Capacidad de seguir cualquier tipo de ducto plano por 30 metros;
- Sensor lateral de distancia;
- Sensores de inclinación;
- Cable de conexión de 30 metros;
- Tamaño 28x14x18 cm;
- Control desde una laptop externa (figura 36).

**Figura 35: el robot en acción**

**Figura 36: el equipo completo**

## 2.3.2 Entre Portugal, Alemania, Reino Unido e Irlanda:

### *Proyecto CORTEX (2001-2004)* [*http://cortex.di.fc.ul.pt/index.htm*]

**Objetivos**

El objetivo principal del proyecto *CORTEX* es explorar los fundamentos teóricos y las cuestiones ingenieriles necesarias para sustentar y validar el uso de objetos sensibles para construir aplicaciones proactivas de gran escala.

**Descripción del trabajo**

Se diseñará un modelo de programación que soporte tal desarrollo de aplicaciones construidas desde objetos móviles incluyendo: a) el desarrollo de un modelo de objetos apropiado y de un paradigma para las comunicaciones, el cual será construido sobre la base de resultados recientes concernientes a comunicaciones de gran escala que dividen en zonas y explotan la percepción de la topología; b) el desarrollo de métodos para expresar propiedades de la calidad del software en el modelo; c) el desarrollo de un modelo global de aseguramiento de la calidad.

También se busca diseñar un modelo de interacción para cooperación entre objetos sensibles y el diseño de una arquitectura abierta y escalable que refleje la estructura heterogénea y la performance de las redes usadas para apoyar al modelo de programación.
Esto conllevará al desarrollo de modelos abstractos de redes para describir las propiedades de las redes subyacentes y al desarrollo de los protocolos y servicios requeridos para sustentar la funcionalidad deseada.

Finalmente se deberá desarrollar una demostración que permita evaluar la tecnología.

**Desafíos fundamentales**

Independientemente del nivel de abstracción en el que se esté trabajando, hay tres problemas fundamentales que deben resolverse para sustentar aplicaciones basadas en objetos sensibles:

- diseminación de la información para crear conocimiento común, mutua conciencia y bases para decisiones locales;
- conseguir la coordinación entre los objetos para llevar a cabo acciones en una forma consistente;
- actuar sobre el entorno, cambiando su estado como resultado de decisiones proactivas o reactivas.

Difundir la información resultante de los sensores o, más genéricamente, de otros objetos es claramente fundamental para las aplicaciones mencionadas anteriormente. Un modelo de comunicación que sustente comunicaciones anónimas y con bajo acoplamiento entre objetos parece obligatorio para soportar la movilidad de los objetos y la evolución de la aplicación. Además, el paradigma de comunicación deberá proveer

la distribución de la información en forma oportuna y confiable ante una variedad de entornos de red y bajo una variedad diferente de cargas.

**Socios del proyecto**

Este proyecto reúne a la Facultad de Ciencias de Lisboa, Portugal (FCUL: Faculty of Sciences of the University of Lisboa); la Universidad de Lancaster del Reino Unido; y al Colegio Trinity de Dublin, Irlanda (TCD: Trinity College Dublin)

## 2.3.3 En Irlanda:

[*http://www.medialabeurope.org/research/anthropos/index.html*]

**Figura 37: Duffy con su creación**

Investigadores irlandeses del MediaLab dirigidos por el Dr Brian Duffy (figura 37) están tratando de hacer que sus robots sean más parecidos a las personas para que sea más fácil la comunicación. Tienen un prototipo llamado *Anthropos* (*Hombre* en griego). Presenta dos cámaras como ojos y un parlante por boca. El proyecto apunta a construir un robot que logre que a la persona se le olvide que está hablando con una máquina. Pero por el momento la conversación termina siendo en un solo sentido ya que Anthropos ha mostrado en la práctica que termina ignorando varias de las respuestas que le son dadas: se trata sobre todo de un robot que hace preguntas y espera un Sí o No por respuesta (aunque bien podría considerarse bastante humano ya que este también es el comportamiento de algunas personas!!).

## 2.3.4 En el Reino Unido:

### *Birmingham* [*http://www.cs.bham.ac.uk/research/robotics/*]

En el Laboratorio de Robots Inteligentes de la Universidad de Birmingham se investigan técnicas para el control de robots móviles. Su trabajo se basa principalmente en aprendizaje y robots evolutivos, aunque también han hecho investigaciones en la parte de navegación.

**Han estado trabajando en las siguientes áreas:**

- Implementaciones de aprendizaje por refuerzo en robots reales;
- Algoritmos probabilísticos para el rastreo de la posición en la navegación ;
- Aprendizaje por Refuerzo en espacios continuos de estados;
- Aprendizaje por Refuerzo con modelos probabilísticos gráficos, especialmente Redes Dinámicas Bayesianas (*Dynamic Bayesian Networks*, DBNs);
- Evolución de la colocación de los sensores;
- Evolución del procesamiento perceptivo;
- Aprendizaje continuo usando redes neuronales constructivas.

El laboratorio está equipado con robots para su uso en investigaciones y enseñanza. Actualmente cuentan con:

- Pioneer y Khepera para investigación;
- Veinte kits de robots para enseñanza e investigación.

## Algunos de los proyectos actuales:

### Cómo actuar mientras aprende

¿Cómo debe actuar un agente mientras está aprendiendo? En este proyecto de Jeremy Wyatt se han desarrollado una serie de algoritmos para encontrar políticas cercanas a las óptimas en procesos de decisión de Markov (MDP). En uno de ellos, seleccionaron un modelo optimista sobre una densidad de otros modelos posibles. Están testeándolo actualmente en más entornos y extendiéndolo a semi-MDPs.

### Estimando la variabilidad en Aprendizaje por Refuerzo

Jeremy Wyatt y su equipo están interesados en aplicar límites a la performance de las políticas diseñadas. Se encuentran desarrollando actualmente una nueva técnica basada en la construcción de intervalos para MDPs. Esto permitirá responder aproximadamente preguntas, como por ejemplo, ¿cuál es la variabilidad probable en la performance para una política dada? y ¿cuál es el peor castigo que puedo esperar tener dada una política?.

### Recombinando modelos de proceso para nuevas tareas

En este proyecto, Jeremy y su equipo están interesados en la idea de poder combinar información de modelos de proceso para separar tareas. Esto permitirá aprender más rápidamente en nuevas labores que compartan algunos aspectos con los modelos para las tareas previas.

### Aprendizaje continuo en Robots

El trabajo previo en aprendizaje en máquinas ha demostrado que la performance de los algoritmos de aprendizaje puede mejorarse transfiriendo conocimiento entre tareas relacionadas. Usar el conocimiento aprendido como fuente de la mejora del aprendizaje puede ser particularmente apropiado para agentes, tales como robots móviles que enfrentan muchas tareas en su tiempo de vida. Axel Grossman y Riccardo Poli estudian la transferencia del conocimiento aprendido usando redes neuronales constructivas[10] que incrementan su capacidad con el número de tareas afrontadas o cuando el entorno del robot se altera.

[10] En el enfoque constructivo se elabora una arquitectura neuronal adaptada a cada problema en particular, la cual aprende crecientemente el número de unidades y pesos sinápticos, utilizando la información contenida en el conjunto de aprendizaje, evitando pruebas y errores en el diseño de la arquitectura. Monoplan y Netlines son dos algoritmos que han mostrado ser muy eficientes dentro del enfoque constructivista, los cuales aprenden mientras crecen, modificando tanto la arquitectura como los pesos sinápticos mediante la regla de aprendizaje llamada Minimerror.

**Localización robusta de robots móviles desde lecturas escasas y con ruido**

En este proyecto, Grossmann y Poli muestran cómo robots móviles con escasos sensores –y con ruido en sus lecturas- pueden emplear la transformada de Hough para hallar rasgos (como paredes), los cuales son más robustos que los aspectos individuales cuando son usados para estimar la posición. El sistema de rastreo de la posición está basado en un mapa conocido y una grilla de probabilidad. Afirman que su método evita los problemas comunes de detección de características en datos del sonar, tales como líneas erróneas a través de agrupamientos separados, inferencia de esquinas y artefactos lineales, a través de la reflexión de la señal emitida. Además sostienen que el método es robusto y computacionalmente eficiente.

## 2.3.5 En Estados Unidos:

### *Carnegie Mellon*[11] [*http://www.ri.cmu.edu/*]

**Figura 38: 'cara' de un robot**

Existen otros investigadores que se preocupan porque el robot tenga un aspecto amigable. En el instituto de robótica de la Universidad Carnegie Mellon están desarrollando un robot con personalidad, que se pueda comportar de acuerdo a las convenciones socialmente aceptables. La idea es mejorar la comunicación que por el momento es deficiente, por ejemplo, las voces sintetizadas no tienen la entonación que las personas usan al hablar. Han pensado que una forma sería tener una cara "humana" que, al hablar, agregue expresiones de acuerdo a lo que dice. Para eso le agregaron a un robot un monitor plano donde se despliega la cara (figura 38). También cuenta con un scanner láser para dirigirse hacia la persona que se encuentra cerca de él cuando le habla.

### *Stanford* [*http://robotics.stanford.edu/*]

En cambio, en la Universidad Norteamericana de Stanford han decidido mejorar la voz sintetizada para darle una personalidad al robot. Se apoyan en que parece ser imposible que el cerebro humano no intente imaginarse "cómo es la persona con la que está

[11] se reitera que estos no son los únicos proyectos de estas universidades norteamericanas, ni siquiera los mejores. Son simplemente algunos elegidos arbitrariamente.

hablando". Aún cuando sepamos que no es una persona y aunque se les recuerde, continuamente, a los participantes de los experimentos que están hablando con una voz sintetizada. Han descubierto que la persona queda perturbada si el tono de la voz no es adecuado a lo que se está diciendo, por ejemplo, usando una voz triste para dar una buena noticia ó una voz alegre para dar una noticia triste; lo mismo pasa si escuchan una voz masculina promocionando algo femenino y viceversa.

### *Brown University*

[*http://www.brown.edu/Administration/George_Street_Journal/vol26/26GSJ21b.html*]

Investigadores de la Brown University en Rhode Island buscan hacer posible manejar una computadora con las ondas cerebrales. Esto sería de enorme valor para los parapléjicos, permitiéndoles manejar cosas con el pensamiento. Por el momento han conseguido hacer que unos monos logren manejar un juego de videogame. Los monos deben perseguir una mancha púrpura moviendo una roja. Al comienzo usan un joystick para mover la mancha roja y luego, se les desconecta el joystick (sin que lo noten) y los monos continúan moviéndola solamente con los pensamientos. Los científicos dicen que esto es posible gracias al electrodo que fue implantado dentro del cerebro de los monos, el cual graba señales desde la corteza motora como si movieran el joystick. Los científicos analizan luego las señales con una fórmula matemática, la traducen y alimentan la computadora donde son reconstruidas como direcciones. Se pretende llegar, en un corto plazo, a manejar un cursor con la mente, lo cual permitirá que un parapléjico pueda realizar tareas tales como navegar por Internet o revisar su correo electrónico; y, a largo plazo, identificar qué señales mueven la mano y ayudarlos, de repente, con un robot que realice determinadas tareas y sea controlado por las ondas cerebrales.

### ***University of Boston*** (Department of Cognitive and Neural Systems CNS)

[*http://www.cns.bu.edu*]

El Departamento de Sistemas Cognitivos y Neuronales proporciona entrenamiento y experiencia en investigación a estudiantes graduados y estudiantes universitarios

calificados. Éstos deben estar interesados en los principios computacionales y neuronales, en los mecanismos y arquitecturas que son la base del comportamiento humano y animal, y en la aplicación de arquitecturas de redes neuronales a la solución de problemas tecnológicos. El departamento se enfoca principalmente en dos cuestiones: ¿Cómo controla el cerebro al comportamiento? Esta es una forma moderna del problema Mente/Cuerpo. Y la segunda es ¿Cómo puede la tecnología emular la inteligencia biológica?. Esta pregunta necesita ser respondida para desarrollar tecnologías inteligentes. Se precisa también la adaptación autónoma a un mundo cambiante para resolver varios de los problemas sobresalientes en la tecnología. Los modelos biológicos han inspirado nuevos diseños para las aplicaciones.

Los estudiantes del CNS son entrenados en una gran cantidad de áreas concernientes a la neurociencia computacional, la ciencia cognitiva y sistemas neuromórficos. El entrenamiento biológico incluye el estudio de los mecanismos del cerebro para la visión y reconocimiento visual de objetos; audición, habla y comprensión del lenguaje; categorización, y memoria a largo plazo; procesamiento de información cognitiva; navegación, planeamiento y orientación espacial; dinámica cooperativa y competitiva para redes y memoria a corto plazo. El entrenamiento tecnológico incluye métodos y aplicaciones en procesamiento de imágenes; múltiples tipos de procesamiento de señales; reconocimiento adaptable de patrones y predicción; fusión de información y robótica.

La base de este amplio entrenamiento es la currícula de 17 cursos de grado interdisciplinarios que se han desarrollado. Cada uno de los mismos integra la información sicológica, neurobiológica, matemática y computacional necesaria para la investigación de los aspectos teóricos fundamentales concernientes a los procesos de la mente y el cerebro; y para las aplicaciones de redes neuronales y sistemas híbridos a la tecnología.

### ***University of Southern California*** (Robotic Research Laboratory)
[*http://robotics.usc.edu/index.html*]

El laboratorio robótico de sistemas embebidos de esta Universidad realiza investigaciones en dos áreas relacionadas:

1. El control y coordinación de un gran número de sistemas distribuidos embebidos
2. El control de sistemas con dinámicas complejas

En la primer área estudian el comportamiento de grandes redes de sensores compuestas por fuentes heterogéneas de información, incluyendo pequeños dispositivos, sensores y robots móviles. En particular, su filosofía es que el comportamiento interesante y útil de los robots depende de un uso inteligente off-board de la percepción y comunicación. Se enfocan sobre sistemas cuyo control en bajo nivel sea “fácil” ya que los desafíos están en la coordinación a gran escala.

En la segunda área, estudian sistemas cuya dinámica hace que el problema del control en bajo nivel sea “difícil”. Ejemplos de esto son los robots que saltan, robots helicópteros, entre otros.

## Proyectos actuales:

- **SCOWR** (Scalable Coordination of Wireless Robots): Coordinación escalable de Robots inalámbricos;
- **MARS** (Mobile Autonomous Robot Software): Software para robots móviles autónomos;
- **AVATAR** (Autonomous Vehicle Aerial Tracking and Reconnasiance): Rastreo aéreo y reconocimiento de vehículos autónomos;
- **DML** (Dynamic Monopod Locomotion): Locomoción Dinámica Monópoda (con un solo pie).

### *University of California, San Diego* (CSE Computer Science and Engineering)

[*http://www.cse.ucsd.edu/Research/ai.html*]

La investigación actual del laboratorio de IA del departamento de CSE está enfocada en problemas de procesamiento del lenguaje natural, visión, razonamiento y especialmente aprendizaje. Muchos de los proyectos involucran el uso de algoritmos genéticos y redes neuronales. Gran parte de esta investigación está vinculada con una comprensión del proceso del conocimiento humano. Por esa razón, se mantiene un vínculo cercano con investigadores de la misma Universidad con experiencia en esas áreas incluyendo al Departamento de Ciencia del conocimiento, Sicología, Lingüística, y Neurociencias. También se han establecido contactos entre la Facultad de IA e investigadores que están trabajando en áreas similares en el Salk Institute [*www.salk.edu*].

## Proyectos actuales:

**Visión**

Tienen proyectos sobre compresión de imágenes, procesamiento de texturas y reconocimiento de caras y de objetos.

**Procesamiento del Lenguaje Natural**

Investigan las propiedades formales del lenguaje natural, procesamiento estadístico del cuerpo de textos largos, resolución de ambigüedades en el sentido de las palabras y aprendizaje desde instrucciones lingüísticas.

**Razonamiento**

Trabajan sobre razonamiento automático con una amplia gama de enfoques, los cuales van desde complejas y teóricas investigaciones sobre la expresividad de la lógica no-monótona a arquitecturas para sistemas expertos.

**Aprendizaje**

Aprendizaje es el tema central de la mayor parte del trabajo en el laboratorio de I.A. Sus esfuerzos incluyen una etapa de desarrollo de algoritmos genéticos y su uso como seleccionadores para arquitecturas de redes neuronales; aproximaciones a redes neuronales para aprendizaje dinámico de comportamientos; métodos estadísticos para la extracción de regularidades sintácticas y semánticas desde un texto, etcétera.

## *MIT (Artificial Intelligence Laboratory)*

## Proyectos de Investigación:

### *Robot fisioterapeuta*

**Figura 39: robot fisioterapeuta**

En el área de la fisioterapia están ayudando a las personas a recuperar la movilidad luego de un ataque al corazón. Investigadores del MIT creen que las víctimas de ataques cardíacos pueden mejorar en un 15% su recuperación. El equipo espera desarrollar el robot terapéutico para que las personas recuperen movimiento en las manos y muñecas. La manera tradicional es que los ejercicios se realicen supervisados por un fisiatra humano, aunque la respuesta a los tratamientos tradicionales es, a menudo, bastante pobre.

En esta nueva terapia el paciente debe manejar un joystick con sus manos y desarrollar diferentes tareas repetitivas que le van implicando varios movimientos de la mano y la muñeca (las tareas son del estilo de mover un cursor hasta situarlo entre dos objetivos, ver la figura39). Lo más novedoso es que si la persona no puede realizar el movimiento, el robot lo ayuda a mover el brazo. Si la persona, en cambio, inicia el movimiento por sí misma, entonces el robot provee solamente un nivel ajustable de supervisión y asistencia para ayudarlo. A diferencia de un fisioterapeuta humano el robot puede guiar al paciente a través del mismo movimiento miles de veces. Más de 18000 movimientos pueden ser realizados en 6 o 7 semanas de tratamiento con el robot. El siguiente paso será desarrollar un guante controlado por robot que ayudará a los pacientes a abrir y cerrar sus manos para mejorar su capacidad de manipular objetos.

### *Lego*

Hay investigadores como Mitchel Resnick que no buscan construir robots que piensen, sino juguetes inteligentes que estimulen a los niños como nunca antes lo hicieron. Lego apoya esta visión brindando Lego Mindstorms, el cual permite construir diversos tipos de robots a un precio accesible (ver capítulo 3.1.1 por más información sobre Lego Mindstorms), Resnick lo ha estado utilizando por años en sus investigaciones.

Considera que las personas aprenden más cuando crean y diseñan cosas y ésta es una forma estupenda de hacer eso mismo. No le parece que los juguetes que hay hoy por hoy en el mercado permitan, en general, que el niño aprenda creando. Confía en que el niño podrá, en un futuro, no solamente construir robots que se muevan por el entorno, sino también comunicarse entre ellos. Así, el niño irá comprendiendo los mecanismos de la comunicación mientras juega. En otras palabras, piensa utilizar los robots como herramientas para desarrollar la inteligencia de los niños.

### *Robots móviles inteligentes y adaptables*

[*http://www.ai.mit.edu/people/lpk/mars/index.html*]

La meta es facilitar a los humanos la tarea de programar robots usando aprendizaje por refuerzo y proveyendo de una descripción de alto nivel de la tarea, en la forma de una función de recompensa en vez de instrucciones explícitas de control. Un proyecto paralelo está desarrollando algoritmos que serán usados para la navegación de agentes móviles autónomos.

### *Biomechatronics*

[*http://www.ai.mit.edu/people/hherr/biomech.html*]

Los músculos permiten responder a distintas cargas de trabajo modulando su estructura para tareas específicas. El músculo puede generar más de 4000kJ de trabajo a partir de solamente 1 Kg de glucosa y permite que las extremidades sean livianas pero fuertes. Una meta a largo plazo de este grupo es diseñar máquinas que exhiban agilidad, fuerza y velocidad en diferentes entornos naturales. Buscan construir un actuador que se comporte como un músculo. Si bien existen aproximaciones como el gel de polímero, todavía no logra tener la velocidad de contracción necesaria para varias aplicaciones.

### *Centro para Aprendizaje Computacional y Biológico*

[*http://www.ai.mit.edu/projects/cbcl/web-homepage/web-homepage.html*]

El Centro para Aprendizaje Computacional y Biológico del MIT estudia el problema del aprendizaje con una aproximación multidisciplinaria en las áreas de teoría, aplicaciones ingenieriles y neurociencia.

### *Lenguajes Dinámicos para Sistemas Adaptables*

[*http://www.ai.mit.edu/projects/dynlangs/*]

El grupo de Lenguajes Dinámicos para Sistemas Adaptables está explorando el diseño e implementación de la próxima generación de lenguajes de programación que sustenten sistemas de software complejo, inteligente y adaptable.

### *Aprendiendo modelos tratables y enriquecidos del mundo real*

[*http://www.ai.mit.edu/people/lpk/ntt/home.html*]

En el MIT también están construyendo sistemas robóticos con un brazo y una cámara (actualmente, en simulación), que puedan aprender modelos relacionales del entorno desde los datos percibidos. Los modelos capturarán la incertidumbre inherente del entorno y sustentarán el planeamiento por vía de muestreos y simulación.

### *Leg Laboratory*

[*http://www.ai.mit.edu/projects/leglab/*]

Es un laboratorio que construye robots con piernas y estudia la capacidad de correr, caminar y los problemas derivados de estas tareas, tales como el balance, la inercia, el impulso, y la dinámica entre otros.

### *Máquinas Vivientes*

[*http://www.ai.mit.edu/projects/living-machines*]

En los últimos 15 años se han construido criaturas artificiales basadas en comportamientos -situadas en el mundo actuando como los insectos- y recientemente se han desarrollado robots que pueden tener interacción social como los humanos. Pero nunca nos olvidamos del todo que esas máquinas no están vivas. Quizás nos esté faltando algo fundamental y actualmente inimaginado en cada modelo de comportamiento, percepción, evolución, o selección natural. Si esto es cierto, necesitaremos tener nuevas formas de pensar sobre los asuntos de las máquinas vivientes si es que vamos a realizar un progreso. Este proyecto es un intento de hallar esos elementos perdidos.

### *Medical Vision Group*

[*http://www.ai.mit.edu/projects/medical-vision/*]

El objetivo del Medical Vision Group es desarrollar nuevos algoritmos para análisis de imágenes médicas y visualización de simbolismos médicos, así como construir un sistema basado en visión para navegación y planeamiento quirúrgico.

### *Grupo de robótica basada en modelos y sistemas embebidos*

[*http://web.mit.edu/embeddedsystems/*]

Este grupo esta desarrollando un nuevo paradigma para crear, rápidamente, colecciones de robots exploradores de larga vida que razonen rápido, extensa y exactamente sobre su mundo. Esos exploradores son prototipados velozmente a través de un enfoque que denominan programación basada en modelos.

### *Open Mind – 1001 Questions (Learner)*

[*http://teach-computers.org/learner.html*]

Es un sitio web interactivo donde una computadora aprende de los usuarios que visitan dicha página. Efectúa preguntas simples y aprende de sus respuestas. Cuanto más sabe, mejor pregunta. El objetivo del proyecto es recolectar el sentido común, que todas las personas poseen, pero del que carecen las computadoras. Con esto se busca facilitar el trabajar con computadoras en el futuro y que ello sea de mayor utilidad.

### *Proyecto Aries*

[*http://www.ai.mit.edu/projects/aries/*]

Las máquinas paralelas de hoy adolecen de varias carencias: pobre programabilidad; inadecuada performance de la red y limitada escalabilidad. El grupo de investigación Aries está explorando la construcción de un novedoso microprocesador, red y software para superar esas limitantes. Está apuntalando la integración procesador/memoria para diseñar un sistema altamente programable y escalable hasta un millón de nodos.

### *Vision Interface Project*

[*http://www.ai.mit.edu/projects/vip*]

Este grupo intenta hacer que las computadoras sean conscientes de sus usuarios y entornos. Sus investigadores están construyendo interfaces perceptivas humano-computadora usando técnicas de visión por máquina integradas con procesamiento acústico y otras modalidades.

**Otras consideraciones sobre el MIT**

Esta institución cuenta con el apoyo de agencias del gobierno de los EEUU (DARPA –Defense Advanced Research Projects Agency- , la ONR –Office of Naval Research-, la NASA y la NIH –National Institutes of Health-). También de empresas privadas como NTT (empresa de telecomunicaciones japonesa), Acer, Delta, Hewlett-Packard, Philips y Nokia. Esto les permite desarrollarse tanto en proyectos de aplicación práctica (con resultados a corto plazo), como también en proyectos de investigación (con *posibles* resultados a largo plazo).

## *Notas*

Como pudimos constatar, algunos de los proyectos de las universidades son auspiciados por el estado (en EEUU principalmente) y otros, por la industria privada. Este apoyo resulta fundamental, sobre todo, dado el elevado costo del hardware en estos proyectos. Es que, a diferencia de otros tipos de proyectos donde el costo está dado fundamentalmente por el software utilizado, en estos casos el software lo estamos diseñando nosotros, por lo que el hardware es el que limita nuestras posibilidades. Además, el apoyo de la industria es importante ya que aporta los requerimientos prácticos para aplicar los conocimientos desarrollados. Hay que tener en cuenta que es posible que los resultados de estas investigaciones sean obtenidos a largo plazo y que, tal vez, varias de ellas no produzcan un resultado favorable sino un aprendizaje acerca de cuáles cosas no funcionan. Sin embargo, la recompensa de lograr "robotizar" alguna tarea es para algunos tan valiosa que vale la pena intentarlo.

En cuanto a la organización de estos proyectos, algunas universidades han optado por encararlos recurriendo a grupos multidisciplinarios, con un enfoque donde se ataca al gran problema y se lo va resolviendo desde el concepto original con varias ópticas y se debate y filosofa al respecto. Otras universidades, en cambio, reducen el problema hasta que resulte manejable por un grupo de expertos informáticos y se plantean resolver esa tarea aislada, luego otra y después las van integrando. Cualquiera de los enfoques es válido y ninguno de los dos ha mostrado ser mejor que el otro al momento de comparar los resultados obtenidos. También se pueden dividir los proyectos siguiendo una analogía al problema del aprendizaje por refuerzo (que vimos en la sección 1.2.4.1) de exploración vs explotación: hay quienes trabajan en la parte de la exploración buscando nuevas técnicas y conceptos teóricos y están aquellos que buscan aplicar (explotar) las técnicas ya conocidas a problemas desconocidos. Al momento de abordar el tema de la robótica, tenemos que elegir en cuál grupo nos vamos a ubicar. En las aplicaciones que desarrollaremos en esta tesis elegimos formar parte del segundo grupo, es decir, no buscamos inventar una nueva técnica sino aplicar una ya conocida a una tarea original.

Existe también un pequeño grupo de personas que se esfuerzan para elaborar un mejor marco teórico de las técnicas, pero no hay, todavía, suficiente investigación en esta línea. Primero porque, a muy pocos les atrae el trabajo que requiere formalizar una técnica (es una tarea que ha estado históricamente más asociada con los matemáticos o físicos que con los ingenieros), y segundo porque la aparición de dificultades al aplicarlas hace dudar sobre la correctitud de las mismas. Estamos, además, en una etapa de desarrollo bastante dinámica puesto que surgen, cada cierto período, nuevos algoritmos o variantes de los anteriores. Lo que también significa que se podrían estar teorizando y formalizando técnicas antiguas que no se usan más porque hay una nueva o una versión mejorada. De todos modos, es necesario tener un marco teórico ya que permitiría identificar componentes claves y mejorar los actuales. Una vez que contemos con un modelo, se podrán inferir reglas y aplicar incluso algunas que conozcamos de otras disciplinas. Siempre es bueno contar con un modelo; el hecho de que el mismo sea incompleto no debiera de desanimarnos ya que esto es así por definición (un modelo es una representación incompleta de algo). Además, resulta ser un punto de partida siempre mejorable a medida que se le van encontrando errores.

# 3. Aplicaciones

Nos planteamos el desafío de programar un robot en dos aplicaciones diferentes. Uno que debe seguir las paredes por el lado derecho para siempre. Y otro que debe lograr arrimar un objeto hasta cierta posición en un escenario donde solamente están presentes el robot y el objeto (no hay paredes, ni otros robots, ni otros objetos). Dicho de una manera más pintoresca: un robot hormiga que debe llevar la comida hasta su nido empujándola.
La primer tarea es útil para otras como resolución de laberintos o navegación. La segunda es útil para aplicaciones, tales como las competencias de fútbol entre robots o las lucha de sumo; también para otras, como la comentada en [Laurent and Piat 2001] de manipular objetos milimétricos (por ejemplo células biológicas) sobre un vidrio debajo de una cámara CCD.

## *3.1 Estudio de posibles robots y simuladores*

En una primera etapa investigamos buscando alternativas sobre la plataforma de hardware que usaríamos para realizar la aplicación. Descartamos la posibilidad de construir nosotros mismos un robot por carecer del tiempo necesario para dicha elaboración a la par de este proyecto. Además, consultamos con el Instituto de Ingeniería Eléctrica por ser los que hasta el momento presentan la mayor experiencia en el tema robótica dentro de la Universidad de la República, pero lamentablemente no pensaban abordar el tema de construir un robot autónomo hasta avanzado el año 2003. Esto nos llevó a buscar robots ya construidos. Contábamos con un kit Mindstorms de Lego por lo que investigamos su versatilidad buscando proyectos alrededor del mundo donde se usara el kit de Lego para proyectos académicos.
También evaluamos la posibilidad de usar al robot Khepera ya que ofrecía una buena relación entre funcionalidad y precio; contando, además, con que uno de los tutores ya tenía experiencia en el uso del mismo.
Otra variante que encontramos fue que cabía la posibilidad de convertir una Palm en un robot y se investigó al respecto.
Finalmente evaluamos simuladores por ser una mejor solución costo-beneficio. A continuación, comentaremos lo investigado y fundamentaremos nuestra decisión final sobre la plataforma elegida.

### 3.1.1 Lego Mindstorms

El Lego Mindstorm presenta un increíble parecido con un proyecto del MIT: el "*MIT programmable brick*" (se puede observar en las figuras 40 y la figura 41). Pero pese a su notable similitud, el Lego fue construido independientemente, no se basa en trabajo del MIT alguno y ambos difieren en cuanto a software y CPU.


**Figura 40: MIT programmable brick**


**Figura 41: Lego RCX Brick**

### *Breves comentarios acerca del Mindstorm:*

El componente principal es el denominado RCX. Tiene tres enchufes para sensores (1,2,3) y tres enchufes para actuadores (A,B,C). También tiene cuatro botones para acceder a distintas funciones (prendido/apagado, etcétera). Posee sensores de luz, de contacto y motores. También forma parte del kit una torre que permite una comunicación por medio de rayos infrarrojos entre el RCX y la computadora (dependiendo de la versión del kit esta torre se conecta a un puerto USB o a uno de los COM). El RCX posee una memoria donde se le instala el firmware y los programas. Existe la posibilidad de sobrescribir el firmware de Lego con otros, por ejemplo, con uno denominado lejOs que permite posteriormente programar en una versión limitada del lenguaje Java (por más información ver [*http://lejos.sourceforge.net*]). El lenguaje de programación es visual y propietario de Lego. Es pertinente recordar que, como comentamos en la sección 1.2.4, en última instancia alcanza con mandar comandos de

bajo nivel a los accionadores para que el robot reaccione como se desea, por lo que una vez conocido el protocolo de mensajes del RCX (que puede hallarse fácilmente en Internet), alcanza con escribir un programa en cualquier lenguaje que envíe los comandos al puerto de donde lee la torre. Para construir al robot deben agregársele al RCX los bloques clásicos de Lego para darle un "cuerpo", es decir, debe armarse una estructura sobre la que se monta el RCX.

Hay un sitio muy interesante en Internet:
[*http://el.www.media.mit.edu/groups/el/Projects/constructopedia*]
donde el MIT maneja su investigación acerca de posibles diseños estructurales usando el Lego. Se busca que uno realice sus propias creaciones más allá de los planos que trae el Lego de fábrica. Ahí se pueden hallar recursos de partes, construcciones "clichés", etc. El sitio está orientado al diseñador y se parece a un manual de mecánica donde muestran cómo armar algunas construcciones útiles. El MIT tiene, además, una competencia denominada "6270" (es el número del curso). Los estudiantes reciben un kit conteniendo partes de Lego, componentes electrónicos, sensores, baterías, motores, alambres y tienen un mes para diseñar, construir y corregir un robot para la competencia. Este robot debe ser autónomo (controlado por la computadora, no puede dirigirse por joystick). El estudiante aprende en el proceso varias y poderosas ideas de la ingeniería, como por ejemplo, feedback, control, y principios del diseño ingenieril.

## *Extensiones encontradas para el Mindstorm de Lego*

[*http://mindsensors.com*]
Dada la limitante de contar con un máximo de tres sensores y tres actuadores se contempló la posibilidad de usar extensiones y se investigaron algunas alternativas.

### *Switch Multiplexer*

El RCX de Lego Mindstorms tiene solamente 3 entradas para sensores. Muchos diseños de robots necesitan más de tres sensores. El Switch Multiplexer permite conectar tres sensores binarios (interruptores, Sensores de tacto, etc.) a una entrada del RCX (ver figuras 42 y 43) dando la posibilidad de contar con un total de 9 sensores para cada RCX. Con unas pocas consultas, un programa puede conocer cuáles switches están presionados. Un cuarto switch se podría conectar, pero enmascarará a los otros cuando sea presionado.

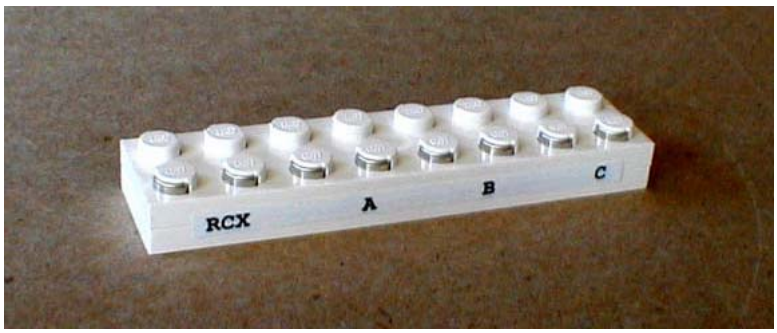



**Figura 42: Switch Multiplexer**

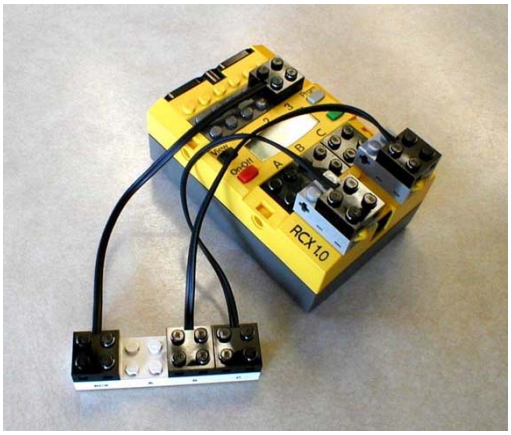



**Figura 43: Conexión al Multiplexer**

El switch multiplexer se programa como el sensor de luz de Lego. Hay que usar el bloque del sensor de luz (*Light Sensor bl*ock) en el entorno de programación de Mindstorms para tomar decisiones (figura 44). El switch multiplexer retorna ocho diferentes valores dependiendo de qué combinación de interruptores están cerrados. En el ejemplo de abajo, el Motor A se encenderá cuando el switch A esté cerrado. El Motor C se encenderá cuando el switch C se cierre. Ambos motores se encenderán cuando ambos switches estén cerrados.

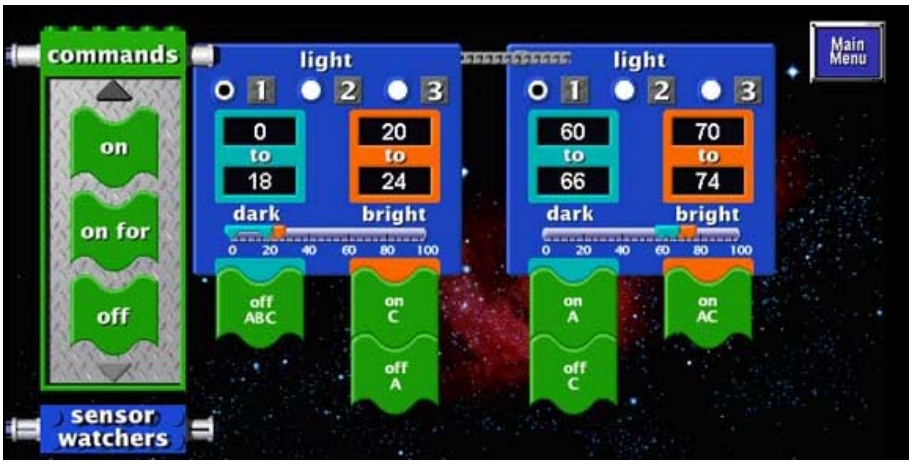



**Figura 44: vista de cómo se programa el switch Multiplexer**

### Bonus switch

Se puede agregar un cuarto interruptor, pero funcionará diferente. Este interruptor está conectado a la misma entrada del RCX que el switch multiplexer. Simplemente hay que apilar el cuarto interruptor encima del cable del multiplexer. Cuando el cuarto switch esté presionado, el valor del RCX será 100, sin importar si los otros switches están o no presionados. Este cuarto switch es útil para indicar un evento que puede ser procesado sin leer los otros switches.

### *Active Sensor Multiplexer: #AMUX3-01*

A diferencia del multiplexor anterior, el Active Sensor Multiplexer permite conectar 3 sensores activos a un puerto de entrada (figura 45). Esto significa que podemos conectar un sensor de luz, o un sensor de inclinación o cualquiera que hayamos construido a un puerto RCX. Puesto que los puertos tienen solamente potencia para manejar a un único sensor, este multiplexor no solamente alterna la señal sino también el suministro de energía del mismo dando la flexibilidad de dos sensores extras conservando la batería.


**Figura 45: Active Sensor Multiplexer**

**Detalles Técnicos**

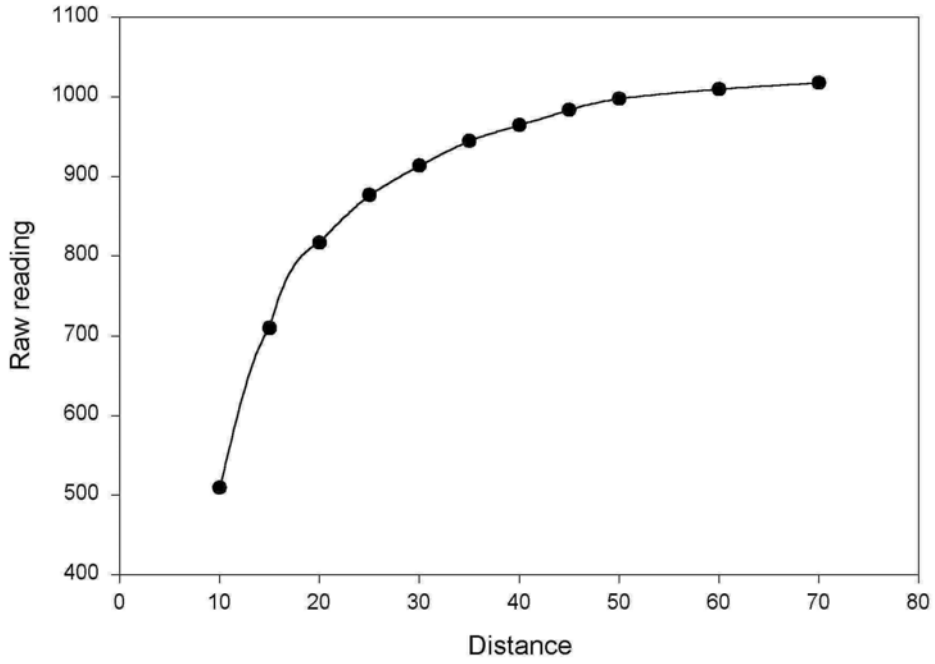
- Consumo de potencia: 4mW (<3% de la energía total disponible);
- Consumo sin sensores conectados: 500 mA;
- Caída de tensión con un sensor de luz conectado: 50 mV;
- Tiempo de selección de canal: 75 ms.

### *Sharp GP2D12 Interface to LEGO RCX*

El sensor infrarrojo (figura 46) puede leer distancias de entre 10 a 80cm. Interconectando el mismo con el RCX de Lego, proporcionamos la capacidad de medir distancias a una pared o un obstáculo, sin embargo, GP2D12 consume más energía de la que un puerto activo del RCX puede proporcionar. Para superar esta desventaja, almacena energía en un capacitor y la usa cuando se ha acumulado suficiente. Este sensor usa un programa para realizar su ciclo de lectura.

**Figura 46: Sharp GP2D12**

**Gráfica de calibración:**
**Use esta gráfica para convertir lecturas de RCX en distancias usando el GP2D12**




Nota: la Calibración puede tener un 10% de variación de sensor a sensor

### *Remote servo motor controller RSC2-01*

RSC2 es un controlador remoto de servomotores. Este servicio recibe comandos del RCX por infrarrojo. Cada RSC2 viene con su propia *byte address*; esto hace posible controlar más de un RSC2 individualmente sin interferencia alguna (figuras 47 y 48) y si se precisa operar varios RSC2 a la vez, puede hacerse enviando a la dirección global en lugar de la de un servicio. (la *Global address* para todos los RSC2 es 0x80)

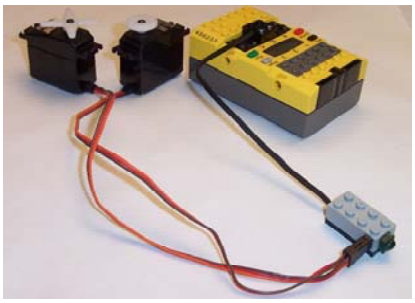


**Figura 47: conexión al controller**

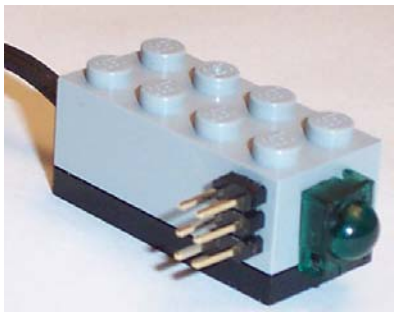


**Figura 48:Remote Servo Motor Controller**

### *Notas*

Debido a que permite conectar un máximo de 3 sensores y a que el tipo de sensor que trae de fábrica es de luz o de contacto, el Lego tiene una limitante importante en su percepción del estado del mundo. La mayoría de los algoritmos precisan de un nivel mínimo de complejidad en los sensores y actuadores en el robot que superan al Mindstorm de Lego. Una solución podría haber pasado por conseguir algunas de las extensiones que mencionamos, pero nos hubiera llevado demasiado tiempo a causa de los pasos necesarios para comprar por Internet debido a la mala infraestructura de los proveedores de esas extensiones para vender fuera de los EE.UU.

Otra desventaja del Mindstorm de Lego es su inestabilidad estructural ya que es bastante probable que se vaya desarmando por los golpes contra obstáculos del entorno. En definitiva, desde su concepción, se trata más de un juguete que de un robot para investigación y se aplica más a proyectos en los colegios que a investigaciones universitarias. Tiene otro enfoque: busca despertar el interés en los robots (nosotros ya tenemos ese interés, buscamos satisfacerlo!!).

Por eso descartamos el Mindstorm y pasamos a investigar lo que aparecía como otra alternativa barata para contar con un robot real: el Palmbot.

## 3.1.2 Palmbot

**Carnegie Mellon Palm Pilot Robot Kit**

Es una carcaza sobre la que se conecta la Palm (ver figuras 49 y 50)

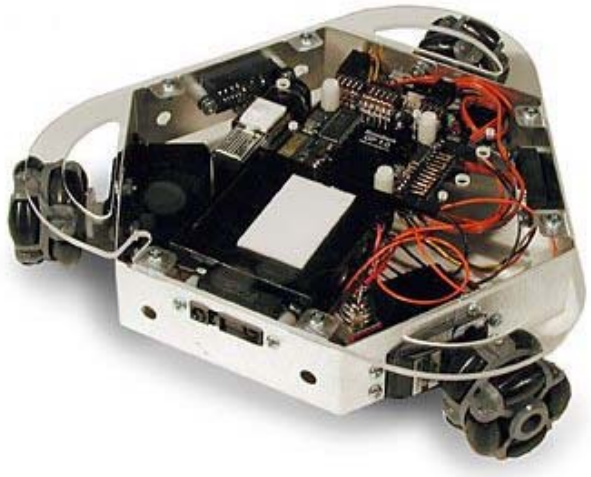


**Figura 49: palmbot**

Aquí presentamos la última versión:

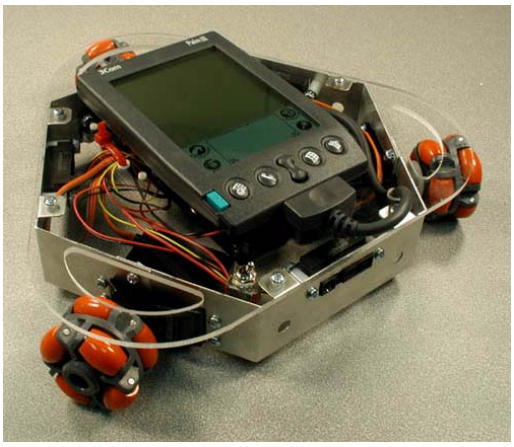


**Figura 50: El kit PPRK (construido en el Carnegie Mellon University Robots Institute)**

**Datos técnicos:**

Viene con un servo-control que se conecta con la palm vía RS-232, tiene la habilidad de controlar 8 servos independientes y 5 entradas análogo-a-digital. El PPRK utiliza 3 de los canales para las ruedas y 3 de las A/D entradas para los sensores de distancia infrarrojos. Las ruedas pueden moverse en dos sentidos a la vez (por ejemplo, puede moverse y rotar al mismo tiempo). En los servos que no son utilizados se pueden agregar pinzas o una brújula. En la página de la Universidad Carnegie Mellon se puede acceder a algunas aplicaciones como PenFollow (permite dibujar en la palm y luego hacer que el robot siga el trazo dibujado).

Encontramos un proyecto que usa dos ruedas en lugar de tres. Para ser más exactos, utiliza discos compactos como ruedas ya que, al ser de mayor tamaño, brindan mayor estabilidad frente a terrenos irregulares y también una mayor velocidad. Para realizar esta modificación, los que realizaron el proyecto tuvieron que modificar los servos para adaptarlos a estas nuevas ruedas. Además, le agregaron una cámara color que usan como sensor de luz.

## *Notas*

Finalmente descartamos esta alternativa pues solamente funcionaba con un modelo antiguo de Palm que teníamos que comprar por separado; además, el costo era similar al del LEGO que ya poseíamos y no era mucho mejor. También teníamos el problema de que nos llevaría mucho tiempo encargarlo haciendo peligrar la finalización del proyecto. Tampoco hay un gran soporte para los usuarios de Palmbot. Si bien existe una lista de correo electrónico de "fanáticos", no hallamos nada que fuese de utilidad entre sus mensajes, incluso observamos anuncios donde integrantes de la lista intentaban vender sus unidades por lo que sospechamos que no son demasiado atractivas. Al mismo tiempo, no está muy difundido como robot para el mundo académico, no hay proyectos interesantes que lo utilicen y nosotros estamos intentando alinearnos con lo que se usa en otras Universidades del mundo.

### 3.1.3 Khepera

Khepera es un robot miniatura producido por la empresa suiza K-Team (ver figura 51). Tiene un diámetro de 5.5cm, una altura de 3.5 cm y un peso de 86 gramos. Posee dos ruedas que permiten que el robot se desplace por el mundo. Cada rueda es manejada independientemente mediante un comando que maneja un valor entre -128 y 127, donde -128 significa a máxima velocidad hacia atrás, 0 es detenerse y 127 es máxima velocidad hacia delante; así, sucesivamente, 1 es lentamente hacia delante, etc. Presenta una velocidad máxima de 60 cm/s y una mínima de 2 cm/s. Tiene ocho sensores infrarrojos que ayudan al robot a percibir el entorno, seis delanteros y dos traseros. Su rango de detección está entre 2 y 5 cms. Los datos del sensor son valores reales entre 0.0 (nada enfrente) y 1.0 (obstáculo muy cercano) y cada sensor es codificado en 10 bits (0 a 1023). Todas las medidas dependen significativamente de varios factores, tales como la distancia hasta el obstáculo, el color, la reflectancia, etc. En la práctica también se comprueba una gran heterogeneidad entre los sensores que refleja el proceso de manufacturación. En cuanto a la autonomía de energía, ésta es de 30 minutos.

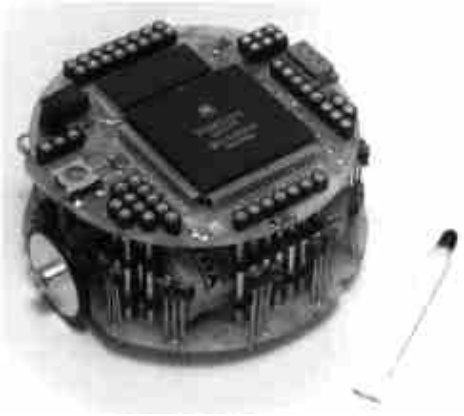


**Figura 51: robot Khepera**

#### *Notas*

Averiguamos en el sitio más cercano donde se vende dicho robot (en Brasil) pero su precio supera los U$S 1800, razón por la cual lo descartamos y pasamos a investigar simuladores que resultaban una alternativa claramente mejor en términos económicos.

### 3.1.4 Simuladores

Finalmente optamos por investigar distintos simuladores gratuitos. Nos centramos en simuladores del robot Khepera y optamos por el más utilizado: YAKS que, además, cuenta con la característica de funcionar sobre Unix y Windows indistintamente[12].

#### *YAKS (Yet Another Khepera Simulator)*

[*http://r2d2.ida.his.se*]

Se trata del simulador "oficial" del robot Khepera. La simulación es bi-dimensional, pero permite observar el mundo desde diferentes cámaras (ver figura 52); también hay versiones en 3-D, pero no son gratuitas. Trae los archivos con los códigos fuente y está programado en C++. Un aspecto negativo es que no cuenta con documentación alguna, por lo que explicaremos aquí lo necesario para entender su uso. Lo principal en la simulación viene dado por el archivo *sim.cpp*, el cual debemos modificar o sustituir para correr la nuestra. Trae un archivo *test.opt* donde se cargan las configuraciones de la simulación (se explicita el archivo donde se describe el mundo donde se mueve el robot, la cantidad y tipo de robots, si se mueven los obstáculos entre simulaciones, etcétera). Una vez generado el ejecutable *yaks.exe* se ejecuta mediante la línea de comando: *yaks test.opt*

El archivo *sim.cpp* que viene con YAKS tiene un código de ejemplo necesario para poder indicar si queremos que se ejecute la simulación por pantalla o sobre el robot Khepera real. Esto permitiría realizar (aunque nosotros lo ignoramos por no contar con el robot real) desde la misma aplicación, la simulación y la comprobación sobre el Khepera. Esta es otra buena razón para elegir a este simulador pensando que, incluso, es la mejor elección a futuro si se logra comprar el robot en el Instituto de Computación.

[12] En realidad todos nuestras pruebas están realizadas sobre Windows, cabe la posibilidad de que la promesa de que nuestras aplicaciones sean portables a Unix no sea del todo correcta debido a incompatibilidades del simulador.

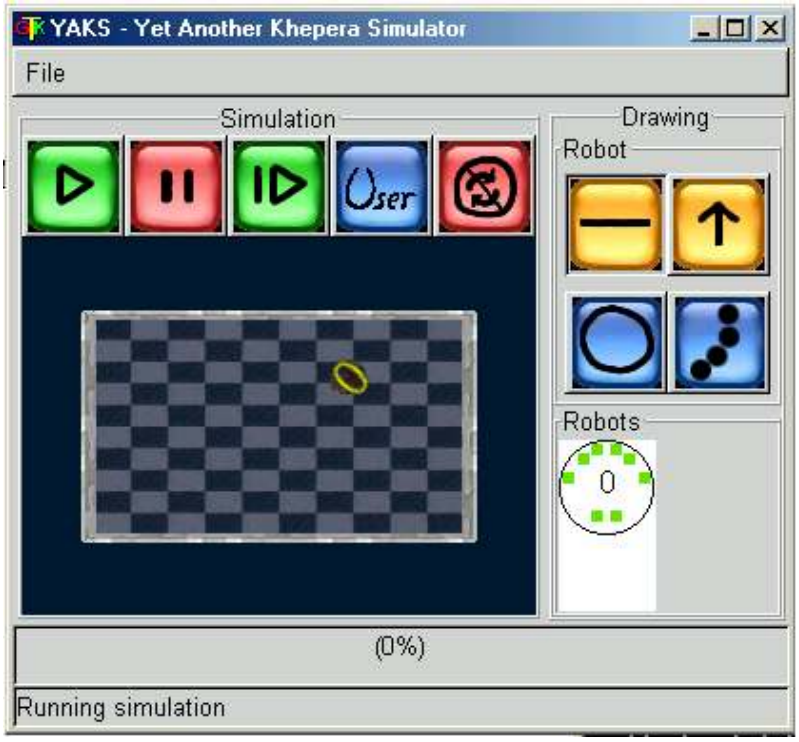



**Figura 52: vista del simulador del robot Khepera, YAKS, en Windows**

**Breve descripción de la Interfaz gráfica de YAKS:**

Como se observa en la figura 52, se cuenta con un conjunto de botones, un área donde se observa la simulación y otra donde se muestran diagramas de cada robot que participa en la simulación con sus sensores variando entre rojo y verde según haya o no lectura en los mismos respectivamente. La ventana cuenta también con un indicador de porcentaje que no viene al caso para nuestros intereses, y con una barra de mensajes en el extremo inferior.

Los botones de la simulación son (de izquierda a derecha): el botón *play* que permite iniciar la simulación; el botón *pause* que permite pausar la simulación; el botón de *un paso adelante* que permite adelantar de a un paso la simulación (no nos queda muy claro la utilidad); el botón *user* que permite, mediante la edición del archivo *gui/callbacks.cpp*, asociar una función personalizada; finalmente, el botón de *no refrescar* que deshabilita la actualización de la interfaz gráfica, lo cual agilita enormemente la simulación aunque no permite observar su evolución, se puede volver a activar el componente gráfico de la simulación volviendo a presionar este botón.

Se cuenta también con un conjunto de botones en la parte derecha de la pantalla que cumplen funciones de dibujo pero, lamentablemente no funcionan en Windows: el botón amarillo muestra/oculta el eje de las ruedas del robot; el otro botón amarillo representa al robot como una flecha; el botón azul muestra/oculta el cuerpo del robot; finalmente, el otro botón azul imprime un trazo del movimiento del robot.

**Variables destacables del Archivo test.opt:**

- **A_#ROBOTS:** cantidad de robots
- **ROBOTD1 [f6b]:** descripción del tipo del robot #1; en este caso, con 6 sensores frontales y también los traseros.
- **WORLD_FILE:** el nombre del archivo que contiene la descripción del mundo en el que se corre la simulación.
- **SCALE:** supuestamente es la escala del mundo, pero en la práctica no observamos que funcionara adecuadamente en Windows.
- **SENSOR_NOISE:** si queremos o no que los sensores tengan ruido para hacerlo más realista.
- **NOISE_PERCENTAGE:** cuánto ruido queremos.
- **MOVE_OBSTACLES:** si queremos que los obstáculos tengan una posición diferente entre simulación y simulación.

**Formato de los archivos con la descripción del mundo:**

Comienza el archivo con una línea donde se enumeran ordenadamente 6 tipos de objetos: la cantidad de paredes, la cantidad de zonas, la cantidad de luces, la cantidad de obstáculos del tipo roundobst y la cantidad de obstáculos del tipo sroundobst.
Luego, al ir estableciendo cada elemento, hay que brindar los valores necesarios. En el caso de los obstáculos, se dan las coordenadas en el mundo y el radio que es el mismo para todos los que son del mismo tipo; para las paredes, se dan las coordenadas desde donde comienza y donde termina cada pared; para las zonas[13], se dan las coordenadas y el radio que puede ser distinto para cada zona; y, para las luces, se dan las coordenadas.

**Funcionalidades que son activadas/desactivadas mediante una <tecla>:**

- <**B**>: "RubberBands", dibuja una línea que vincula al robot con la zona a la que pertenece. Esto sirve por ejemplo para el robot-soccer donde permite saber a qué equipo pertenece cada robot.
- <**C**>: "Camera", cambia la perspectiva de la observación en la simulación.
- <**F**>: "FloorLights", prende/apage las luces que fueron definidas en el mundo donde transcurre la simulación.

[13] Zona es un tipo de elemento que se puede definir en el mundo, tiene como característica interesante que permite detectar cuando el robot entra a la zona y consultar la cantidad de veces que un robot ingresó a dicha zona

- <**L**>: "Lighting", ilumina todo el escenario de la simulación o lo restringe a lo mínimo necesario para poder observar al robot.

- <**R**>: "Reflection", crea un piso espejado.

- <**T**>: "Trail", mantiene el dibujo de las últimas posiciones del robot, es como si el robot se moviera tan rápido que uno lo ve en dos o tres posiciones de la secuencia de movimientos a la vez.

- <**I**>: "Identities", muestra encima de cada robot el número que lo identifica, permite diferenciar en casos donde tenemos varios robots cooperando.

- < **+** >: aumenta el alfa del piso. Afecta al color y al brillo del piso.

- < **-** >: disminuye el alfa del piso. Afecta al color y al brillo del piso.

**Problemas encontrados:**

- Inicialmente, cuando uno intenta compilar los archivos obtenidos de Internet, no compilan adecuadamente: se arregla fácilmente comentando o eliminando las líneas que dan error.
- No se pueden elaborar mundos sin paredes, por lo menos debe contar con dos, de esta manera se fija el largo y el ancho del mundo.
- La clase GA no es lo que parece. Tuvimos intención de implementar algo mediante algoritmos genéticos y quisimos utilizar la clase GA que viene con el simulador, pero no es la implementación de un algoritmo genético sino que algo relacionado con algunas cosas que necesita la clase ANN (redes neuronales).
- En el archivo *sim.cpp* del simulador, el robot se comporta de forma tal que al chocarse contra una pared reaparece en el centro del mundo, para darle mayor realismo modificamos el código para que continuara en la posición más cercana a donde se chocó.
- El simulador no incorpora el comportamiento de empujar objetos, lo tuvimos que programar nosotros. Esto incluye una modificación en algunos de los archivos fuente que componen el simulador, en particular *sim.cpp* (donde se codifica la simulación) y en *environ.cpp* (donde se maneja la interacción con el entorno, se implementan funciones como el chequeo de colisiones y generación de lectura en los sensores, entre otras) ya que la función que movía al objeto lo hacía inadecuadamente y por momentos el robot atravesaba la estructura del objeto quedando incrustado.

## *3.2 Desarrollo de aplicaciones*

Para programar aplicaciones utilizaremos el algoritmo de aprendizaje por refuerzo denominado *Q-learning*. En vez de usar la función de valor[14], *Q-learning* utiliza una *Q-función* que la aproxima. El algoritmo no precisa un modelo del mundo y almacena el valor de refuerzo esperado con cada par estado-acción usualmente en una tabla de búsqueda. Hay tres funciones involucradas: memorización, exploración y actualización.

Algoritmo *Q-learning*: (adaptado de la versión aparecida en [Touzet 1997])

1. Inicializamos la memoria del robot: para todos los pares estado-acción el valor asociado es 0 (i.e., $Q(i,a)=0$).
2. Repetir :
    a. Sea *i* el estado actual del mundo
    b. De acuerdo con algún criterio exploración vs explotación se decide o bien explorar: se elige una acción *a* al azar
       ó explotar: la función de evaluación selecciona la acción que maximiza la recompensa esperada que cumple: $a = \text{argMax}(a)\ (Q(i,a'))$
       donde *a'* representa cualquier acción posible en el estado *i*
    c. El robot ejecuta la acción *a* en el mundo. Sea *r* la recompensa (puede ser nula) asociada con la ejecución de la acción *a* en el mundo.
    d. Actualiza la memoria del robot:
       $Q_{t+1}(i,a) = Q_t(i,a) + \beta.(r + \gamma.\ \text{bestNewQval} - Q_t(i,a))$
       Donde $Q_t$ es el *qvalue* para el instante de tiempo t, *r* es la recompensa, bestNewQval es el mayor *qvalue* para cualquier acción que se tome en el nuevo estado que se alcance luego de realizar la acción *a*, $0<\beta, \gamma<1$. Se puede ver a $\beta$ como la tasa de aprendizaje y a $\gamma$ como una tasa de interés o como cuanto del estado siguiente se comunica al previo (cuanto se propaga hacia atrás).

En la implementación del algoritmo utilizamos un generador de números aleatorios que controla la aleatoriedad de la selección de las acciones al explorar. El criterio exploración vs explotación que utilizamos es el siguiente: tenemos un umbral y sorteamos un número al azar. Si ese número es mayor al umbral, explotamos eligiendo aquella acción que tiene asociado el mayor valor, si no lo es, exploramos eligiendo una acción al azar. La probabilidad de exploración disminuye proporcionalmente al número de visitas al estado actual.

A continuación describiremos cómo enfrentamos las dos tareas: la primera, programar un robot para que siguiera las paredes por la derecha para siempre. Y la segunda, obtener un robot que, a semejanza de las hormigas, buscara una pieza de "comida" y la llevara hasta su "nido" pudiendo únicamente empujarla (no es posible agarrarla con pinzas).

[14] Por más detalles leer la sección 1.2.4

### 3.2.1 Seguir paredes por la derecha

El objetivo es que el robot aprenda a moverse paralelo a la pared siguiéndola por el lado derecho del robot. Este problema fue elegido por su simplicidad y porque, sin embargo, plantea aspectos tales como el entorno variable y dinámico, los cuales son desafiantes para cualquier aplicación con robots. Además, si bien el problema es simple en su enunciación, sobre él descansan problemas más complejos, como por ejemplo, la resolución de laberintos, elaboración de mapas y navegación.

Con el fin de mostrar en detalle el proceso de diseño de una solución es que explicaremos ampliamente los pasos que fuimos dando para resolver esta tarea. La parte más provechosa es la de constatar cuáles fueron los aspectos que afectaron más a la solución encontrada, cuáles causaron mayores complicaciones (y sobre todo errores) y, también, cómo algunas decisiones tempranas de diseño afectaron fuertemente el resto del proceso.

#### *Desarrollo*

Empezamos con la definición de los estados, como se muestra en la figura 53, decidimos simplificar y agrupar los sensores quedándonos 3 grupos:

**Figura 53: agrupamiento de sensores (versión 1)**

De esta manera trabajamos con: *sensorIzquierda* (es igual a $sensor_0 + sensor_1 + sensor_2$), *sensorDerecha* (es igual a $sensor_3 + sensor_4 + sensor_5$) y *sensorTrasero* (es igual a $sensor_6 + sensor_7$).

Definimos 4 estados posibles:

0. No hay lectura en los sensores
1. Sensor Izquierdo presenta lectura
2. Sensor Derecho presenta lectura
3. Sensor Trasero presenta lectura

En un primer intento, detectamos problemas porque los estados definidos por nosotros no eran disjuntos. Por ejemplo, en las esquinas puede haber lectura en los sensores traseros y derechos y dependía del orden en que se evaluara para desempatar el estado en el que se encontraba el robot. Por eso pasamos a definir los estados de manera que tuviera que haber una lectura. Ésta debía ser mayor que un umbral, y en caso de empate el sensor que tuviera el mayor valor marcaba el estado.

Luego definimos (diseñamos) la función de refuerzo. Decidimos hacerla según el estado al que nos llevaba dicha acción desde un estado dado. Se premia o castiga a una acción en base a la transición entre estados (de acuerdo al estado en el que estábamos y el

estado al que llegamos), por lo cual podemos explicitarla sobre un diagrama similar al mostrado en la figura 54.

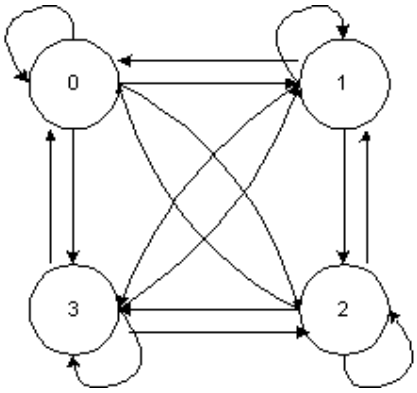



**Figura 54: diagrama de estados y sus transiciones (versión 0)**

Debido a que buscábamos que el robot siguiera las paredes por su lado derecho es que premiamos la llegada y la permanencia en el estado 2; castigamos el abandonar al estado 2; en el resto de los casos simplemente no hacemos nada (dimos recompensas nulas). Esto está representado en la figura 55.

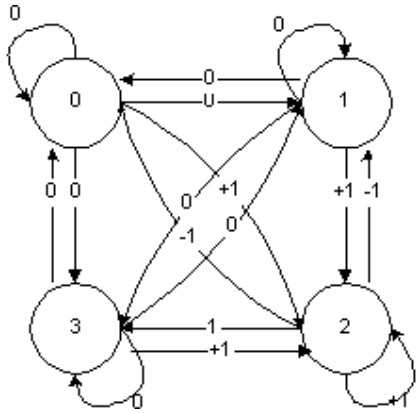



**Figura 55: estados y transiciones (versión 1)**

También agrupamos las acciones que se traducen en diferentes configuraciones de potencia en los motores y simplificamos en 4 posibles movimientos:

1. Ir hacia delante;
2. Girar hacia la izquierda;
3. Girar hacia la derecha;
4. Ir hacia atrás.

**Resumen de los primeros ensayos:**

El primer caso funcionó mal porque el robot, al arrimarse a una pared, quedaba oscilando entre dos posiciones. Esto sucedía debido a que lo premiábamos simplemente por continuar presentando lectura en el sensor de la derecha.

Buscando que el robot encontrara más rápidamente la pared, decidimos darle 100 unidades de premio al arrimarse y 1 unidad de castigo cuando la perdía por alejarse. El comportamiento obtenido fue tal que en lugar de encontrar una pared y mantenerse cerca –recibiendo 1 unidad de recompensa- prefería arrimarse y alejarse porque cada "arrimarse y alejarse" le daba 99 unidades de premio. De esta manera confirmamos que, si uno no define bien la función de refuerzo, es probable que el robot encuentre un comportamiento con una buena recompensa y que no sea el que teníamos en mente al momento de diseñar dicha función. Vale aclarar que nosotros queremos que **avance** paralelo a la pared.

Posteriormente decidimos castigarlo con 100 (se puede ver como una recompensa negativa de -100) si perdía la pared y obtuvimos un robot que disfrutaba "refregarse" en las esquinas y quedaba oscilando. Con este comportamiento, de hecho, se movía sin perder la pared, aunque no era exactamente lo que buscábamos (parecía que nos estuviese tomando el pelo!!).

En otra corrida, donde modificamos la potencia de los motores para las acciones de girar -con el consiguiente cambio de velocidad en el giro a la izquierda y a la derecha-, observamos que, por virar demasiado, atravesaba toda la habitación. Esto ocurrió porque no le permitimos regular cuánto giraba, simplemente definimos una acción "girar a la izquierda" y otra acción "girar a la derecha" sin graduación.

Volvimos a bajar la velocidad de giro. Penalizamos todo movimiento que no fuese premiado (sustituimos todos los ceros de la figura 55 por –1) intentando que aprendiera más rápido. Observamos que en cierto momento se pegaba a la pared, pero con la peculiaridad de que la seguía marcha atrás. Aunque luego, como seguía aprendiendo, cambiaba de comportamiento y quedaba oscilando en las esquinas yendo hacia atrás y hacia delante.

Entendimos que, como premiábamos al robot por presentar lecturas en alguno de sus **3** sensores de la derecha, era complicado para el mismo aprender que queríamos que se moviera paralelo a la pared porque, cuando enfrentaba sus sensores a la misma, quedaba en un ángulo de aproximadamente 45º con ella. Es decir, el robot no quedaba paralelo a la pared y al avanzar o retroceder terminaba casi siempre chocándose o alejándose de ella. Entonces, optamos por cambiar y agrupar a los sensores como se muestra en la figura 56.

**Figura 56: agrupamiento de sensores (versión 2)**

Ahora teníamos 5 estados: *sin lectura* en los sensores, *izquierda*, *derecha*, *mderecha*, y *atrás*; 4 acciones: *giro a la izquierda*, *giro a la derecha*, *avance* y *retroceso*. Y

premiábamos al robot cuando enfrenta al sensor de más a la derecha (estado *mderecha*) ya que era el que determinaba el estado paralelo a la pared.
Seguidamente y luego de estos cambios, el robot aprendió que

- ante ninguna lectura (un valor menor que el umbral en todos los sensores), debía moverse hacia delante
- Ante una lectura en los sensores de la izquierda, debía moverse hacia la izquierda. (Esto es adecuado ya que está buscando enfrentar el sensor de más a la derecha)
- Ante una lectura en los sensores de la derecha, debía moverse hacia la izquierda. (Esto es adecuado ya que está buscando enfrentar el sensor de más a la derecha)
- Ante una lectura en el sensor de más a la derecha, debía moverse hacia delante. (Esto es adecuado ya que lo hace avanzar paralelo a la pared)
- Ante una lectura en los sensores traseros, debía moverse hacia delante.

De esta manera, logramos el comportamiento que buscábamos. Concluimos que un buen agrupamiento de los sensores y, por lo tanto, una buena definición de los estados, puede afectar más a la solución obtenida, que un cambio en los valores de las recompensas. Nuestro error estaba en el agrupamiento, no en la función de aprendizaje por refuerzo.

Modificamos el agrupamiento de los sensores para dar mayor robustez a la solución hallada evitando depender tanto de un solo sensor y agrupamos de 2 en 2 los sensores (ver figura 57).

**Figura 57: agrupamiento de sensores (versión 3)**

El resultado fue que, cuando comenzaba en la mitad del escenario, quedaba oscilando porque aprendía que todas las acciones eran igualmente neutras. Era natural que ocurriera; fuimos afortunados al no pasarnos esto con anterioridad. Lo solucionamos agregándole un comportamiento inicial (instinto) para que avanzara hacia delante y, luego de hallar una pared, que comenzara a aprender.

Por supuesto que el resultado no varió mucho ya que lo único que eliminamos fue el problema inicial de quedarse girando en el medio. Pero con esto no evitamos que, luego, en algún momento dado, pueda perder la pared y quede girando en el medio. Se habría evitado si hubiese aprendido que hay una opción mucho mejor que las otras si no percibe pared alguna ó, si hubiésemos programado un instinto haciendo que el robot siguiera alguna serie de movimientos que siempre lo ayudaran a llegar hasta una de las paredes (de ser posible la más cercana). Esto es complicado ya que no hay una táctica simple para que se arrime, en cualquier escenario posible, a la pared más cercana y debemos evitar que atraviese todo el escenario porque buscamos que no se mueva lejos de las paredes (podemos suponer que estamos entrenando al robot para moverse por acantilados por lo que, el atravesar la habitación, equivaldría a caerse al vacío).

Observando el gran problema ocasionado por los diferentes agrupamientos de sensores, decidimos probar con una versión sin agrupar[15]. De este modo, obtuvimos un total de nueve estados (ocho corresponden a cada uno de los ocho sensores y el noveno al estado de no-lectura en ninguno de los sensores). Al comienzo, premiábamos sólo por una lectura en alguno de los sensores del sector derecho del robot, por lo tanto, aprendió nuevamente a restregarse contra la pared. Valiéndonos de la experiencia ya adquirida pasamos a definir una función en la cual premiábamos la lectura del sensor de más a la derecha, con ello logramos un robot que se arrimaba a la pared y se movía poco. Seguidamente, modificamos la función de recompensa de manera que el premio fuera proporcional a la distancia que se desplazaba mientras continuaba presentando una lectura en el sensor de más a la derecha. Con esto logramos que el robot aprendiera a moverse siguiendo la pared por la derecha. El único problema que se mantuvo fue que no se comportaba adecuadamente cuando perdía la pared porque, como comentamos anteriormente, en el estado de no-lectura en los sensores no había una única acción posible que le garantizara una recompensa. Por esa razón y, aunque sabíamos que no había una forma genérica de hallar siempre una pared, sí existía una heurística para los casos donde el robot la pierda. Basándonos en la pretensión de que el mismo siga la pared por la derecha, optamos por definir al instinto como: *girar a la derecha y avanzar*. De este modo logramos que el robot reencontrara la pared una vez perdida; este instinto también funcionaba al principio de la simulación debido a que lo probábamos en un mundo rodeado por paredes, por lo que tarde o temprano el robot encuentra alguna de ellas. En el comportamiento obtenido había cierta oscilación hacia delante y hacia atrás, por ello decidimos eliminar la acción *moverse hacia atrás*. De esta manera logramos un resultado más parecido a lo que el sentido común nos indica sobre lo que es moverse cerca de la pared. Sin embargo, resultó curioso observar que el primer comportamiento obtenido fue como se muestra en la figura 58.

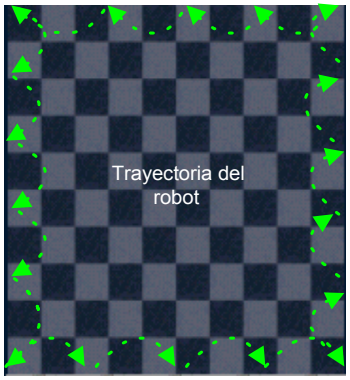



**Figura 58: Trayectoria 'a saltos'**

Tal comportamiento se debió principalmente a la velocidad de giro y a que había aprendido a buscar la lectura en el sensor de más a la derecha girando en sentido horario. Esto significaba que, si algún sensor obtenía una lectura, entonces, el robot se ponía a girar (en sentido horario) hasta que obtuviera otra en el sensor de más a la derecha. Una vez detectada la lectura en el sensor de más a la derecha el robot avanzaba puesto que eso era lo que había aprendido. Hay que observar que no había quedado en una posición paralela a la pared sino algo inclinado y un poco enfrentado hacia el centro

[15] Nótese que sigue existiendo un agrupamiento en la lectura de los sensores, pero éste es más fino que el anteriormente utilizado.

del mundo, por lo que comenzaba a avanzar hasta que perdía la pared. En ese momento giraba hacia la derecha buscando la pared y avanzaba hasta casi chocarse, y de nuevo, comenzaba a girar en sentido horario dando lugar a la secuencia que se puede observar en la figura 58. Para evitar este extraño movimiento decidimos disminuir la velocidad de giro en las acciones *girar a la izquierda* y *girar a la derecha* buscando lograr un movimiento más suave. Así fue como finalmente logramos lo que buscábamos: un robot que, una vez encontrada una pared, la sigue por su derecha (la derecha del robot). Igualmente de vez en cuando tiene alguna dificultad para resolver la situación de doblar en las esquinas ya que percibe una lectura en el sensor de más a la derecha y trata de avanzar sin poder por la pared que tiene enfrente, de todas maneras, tarde o temprano, lo logra y continúa avanzando correctamente.

## *Discusión*

Vimos que -de todas las configuraciones que probamos- la única útil se basó en usar el sensor de más a la derecha ya que éste nos permite determinar el estado paralelo a la pared, hecho fundamental para la solución del problema de seguir la pared por la derecha. Hicimos algunos cambios en el valor de β en la fórmula de *Q-learning,* pero solamente afectó la rapidez con la que se obtiene un comportamiento y el grado de sensibilidad a los cambios del entorno. Hay que notar la no-tolerancia a fallas que supone esta solución por depender de un único sensor. No obstante, cuando intentamos agrupar los 2 sensores de más a la derecha para dar una mayor robustez, no obtuvimos el comportamiento buscado. Una mejor solución sería utilizar otro robot con más sensores en el costado derecho ó modificar al Khepera de modo que pueda girar sus ruedas para enfrentar sus sensores frontales contra la pared y avanzar "de costado". Esto traería también aparejado una mayor complejidad ya que, al momento de definir las acciones, éstas no serían simplemente la potencia de los motores, sino que también deberían incluir el ángulo de giro de las ruedas. Existen también otros problemas inherentes a esta particular distribución de los sensores presentada por Khepera, por ejemplo, notamos que para los sensores traseros, no resulta fácil aprender qué acción tomar dado que si gira, pasa a perder por un momento la pared debido a la separación en la distribución de los sensores alrededor del cuerpo del robot, y en ese momento, interviene nuestro instinto para "recuperar paredes". Esto no ocasiona problemas ya que nuestro instinto funciona adecuadamente para este caso también. Quizás pudiera solucionarse no habiendo agrupado las acciones y permitiéndole elegir un giro con una velocidad mayor tal que le permita pasar de una lectura en un sensor trasero a una lectura en uno de los sensores laterales.

En los ensayos previos a la eliminación de la acción *moverse hacia atrás* pudimos observar oscilaciones respecto a la dirección del movimiento hacia delante y hacia atrás ya que no penalizábamos el cambio de sentido y ambos movimientos tienen cierta similitud. Esto hizo que, por momentos, el robot quedara como si estuviese "rebotando" mientras lo premiábamos. De hecho, la acción de moverse hacia delante y hacia atrás paralelo a una pared era uno de los comportamientos que le daba al robot el mayor premio posible. Observamos en algunos experimentos que el robot podía pasar de comportarse avanzando junto a la pared a moverse marcha atrás junto a la pared, oscilando entre dos comportamientos que maximizaban nuestra función de recompensa.

Como se pudo ver en este caso, no es la cantidad de la recompensa lo que más afectó a la solución, sino la manera en que definimos los estados sobre la base del problema a enfrentar. Esto puede realmente complicarse al tener que expresar metas más complejas con un gran conjunto de sensores ya que, para las personas que deben diseñar los estados, es más simple manejarse con valores simbólicos que con medidas sobre las lecturas de los sensores. También es crítico definir cuándo se dan recompensas y cuándo castigos; inicialmente quisimos definir recompensas menores por cambios de estados que marcaran un avance hacia una lectura del sensor de más a la derecha, por ejemplo, dábamos una recompensa si el robot pasaba de un estado donde predominaba una lectura de un sensor de los de la izquierda del mismo a otro más a la derecha y más cercano al sensor que define el estado paralelo a la pared. Pero no logramos que el robot distinguiera que esas recompensas no eran porque ya hubiera logrado aprender lo que buscábamos que hiciera. De este modo enlentecimos el aprendizaje, pues el robot elegía alguno de los varios comportamientos subóptimos que habíamos definido y demoraba en encontrar el óptimo. Creemos que hay que tener cuidado al inducir al robot mediante recompensas por submetas buscando acercarlo a lo que deseamos. Cuando aumentamos la cantidad de comportamientos que recompensamos, pero que no son exactamente el que buscamos, en algunos casos ayudamos al proceso de aprendizaje y, en otros, entorpecemos la tarea del robot para encontrar aquel que maximiza su recompensa.

Como pudimos constatar, el comportamiento del robot -luego de nuestra etapa de aprendizaje- no es perfecto; además, no siempre el comportamiento que maximiza la recompensa a lo largo del tiempo es el esperado por quien diseñó la función. Una posible mejora consistiría en realizar modificaciones a la función que controla el comportamiento del robot para introducir restricciones al comportamiento observado. Por ejemplo, como disfrutaba refregándose en las esquinas y no era lo que deseábamos, entonces habría que intentar desalentar este comportamiento. Estas restricciones son imposibles de imponer en la función de *q-learning* porque hay que tener en cuenta largas secuencias de acciones y en *q-learning* se maneja solamente la noción de un estado y su siguiente. [Touzet 1997] propone incluir un módulo externo que contenga las secuencias de acciones prohibidas, al estilo de un "código de leyes" donde tendríamos que explicitar -una por una- todas las excepciones que satisfacen a nuestra función de refuerzo, pero no son las que queremos. Por ejemplo, en nuestro caso podríamos prohibir los movimientos hacia delante y hacia atrás oscilando, y el movimiento continuo hacia atrás (en realidad este movimiento no tiene nada de malo pero no es natural para nosotros que el robot se mueva siempre marcha atrás). Este módulo intervendría suprimiendo la elegibilidad de ciertas acciones en una situación particular y en un contexto histórico dado.

De todo esto resulta claro que tener una buena implementación de *Q-learning* puede resultar insuficiente para una aplicación. Además, hay que tener en cuenta que tampoco pueden resolverse todos los problemas del mundo con aprendizaje por refuerzo. En particular, la tarea que nos planteamos puede solucionarse también utilizando un comportamiento reactivo, por ejemplo, haciendo que una vez que el robot encuentre una pared por la derecha, la siga derecho y, si la pierde, entonces doble hacia la derecha hasta encontrarla de nuevo y siga derecho mientras en el sensor de más a la derecha percibiera una lectura mayor a un cierto umbral. Si presenta una lectura en los sensores del frente, entonces que gire hacia la izquierda buscando enfrentar el sensor de más a la derecha.

El entorno en el que realizamos nuestros experimentos es estacionario y la probabilidad de recibir señales específicas de refuerzo no cambia a lo largo del tiempo. Pero el método descrito es eficiente también en entornos no estacionarios que varíen lentamente, sobre todo si no se detiene el aprendizaje y se deja un umbral para que, con cierta probabilidad, se explore (se elija una acción al azar).

Si bien esta tarea (la de seguir paredes) es un clásico de la robótica y es común que, al presentar una nueva técnica, se use como problema simple a ser resuelto con ella, desarrollamos toda la aplicación creyendo que la idea de resolverla usando aprendizaje por refuerzo era original ya que todos los ejemplos de *q-learning* que encontramos mencionaban la tarea de evasión de obstáculos (otro clásico). Luego de finalizar nuestro análisis encontramos que en [Santos and Touzet 1999] aparece una definición de una función de RL para seguir la pared por la derecha. Los autores comentan que ellos tampoco conocen otro paper donde este problema sea resuelto exclusivamente por técnicas de aprendizaje por refuerzo. Esto hace pensar que nuestro intento sería el segundo en hacerlo.

**Función de Refuerzo de Santos y Touzet:**

$RF(s^t,s^{t-1})= +1$ si $g_1(s^t,s^{t-1})< \theta_+$; $RF(s^t,s^{t-1})= -1$ si $g_2(s^t,s^{t-1})< \theta_-$; $RF(s^t,s^{t-1})= 0$ para los otros casos. Donde $g_1(s^t,s^{t-1})= | s_6^t - s_6^{t-1} |$ y $g_2(s^t,s^{t-1})=s_6^t$ .

En la definición se observa que el robot debe ser premiado si se mueve paralelo a la pared (usa el mismo sensor que nosotros para detectar el paralelismo a la pared) y castigado si se aleja de la zona donde su sensor de la derecha puedan percibirla. La solución es dependiente de los valores de $\theta_+$ y $\theta_-$ y no resulta trivial encontrar cuáles son los mejores. Utiliza el método UPA (*Update Parameters Algorithms*) para establecer $\theta_+$ y $\theta_-$ buscando balancear la proporción de experiencias positivas y negativas durante el aprendizaje, esto igualmente no es medida de la performance y en algunos casos el comportamiento obtenido no es muy bueno. También programa dos reflejos: 1º si no percibe lecturas en los sensores, entonces rota 20º en sentido antihorario y avanza; 2º si ocurre una colisión contra la pared, entonces deshace el último movimiento de avance.
Nuestra RF también se centra en el sensor 6, pero no en la magnitud ni en el cambio de valor, sólo es importante que el robot mantenga una lectura mayor a la de los otros sensores. Y, si la mantiene, damos una recompensa proporcional a la distancia recorrida, propiciando el desplazamiento (Santos no lo hace pues se apoya en que sus acciones ya incluyen *avanzar*). En cuanto a los castigos, castigamos -al igual que Santos- cuando pierde la lectura en dicho sensor. Quisiéramos aclarar que nuestro trabajo no se basa, en lo absoluto, en este paper sino que, al contrario, fue desarrollado independientemente. No nos resulta simple comparar su solución con la nuestra ya que los gráficos que aparecen en su paper apuntan a mostrar la evolución de ciertos parámetros propios del algoritmo UPA, pero podemos afirmar que el resultado es, en ambos casos, el buscado: el robot sigue adecuadamente las paredes.

Igualmente, buscando comparar la performance en alguno de los escenarios que aparecen en [Santos and Touzet 1999], reprodujimos el escenario que se muestra en la figura 59.

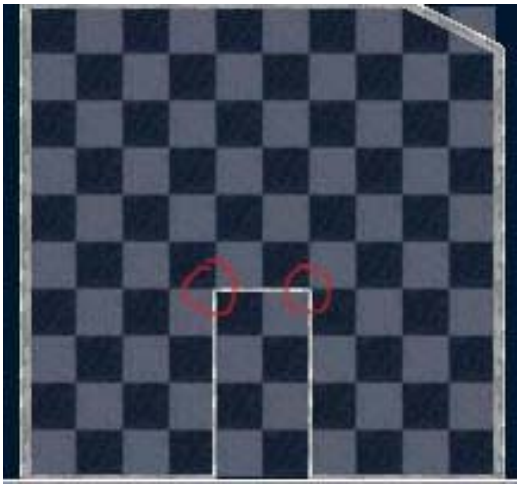


**Figura 59: escenario para testeo del algoritmo**

Pudimos constatar que, debido a nuestro instinto, cuando el robot perdía el contacto con la pared en cualquiera de los dos sectores que marcamos con rojo en la figura 59, el robot giraba un poco hacia la derecha y continuaba en línea recta atravesando el escenario hasta encontrar otra pared. Para corregir esto y hacer que el robot siguiera mejor la pared más cercana a él, modificamos el instinto haciendo que al comenzar la simulación avanzara hasta encontrar una pared, pero luego, si llegaba a perderla, recurríamos a un instinto similar al de Santos (un giro y un avance hasta que reencuentre la pared). De este modo logramos solucionar el inconveniente hallado y obtener un buen comportamiento en dicho escenario también.

Estas experiencias -donde los comportamientos obtenidos variaban según nuestra manera de agrupar- nos hicieron pensar que para los casos donde el número de sensores sea considerable y resulte compleja la tarea de agrupar, podría agregarse al aprendizaje por refuerzo una fase previa con un **algoritmo genético** donde se pruebe con qué agrupamiento de los sensores aprende mejor el robot. En nuestro problema es indudable que lo importante está en el sensor de más a la derecha ya que nos permite detectar que el robot está paralelo a la pared derecha, pero para casos más complejos podría usarse una población inicial con distintos agrupamientos y luego quedarnos con alguno de aquellos que obtuvieron los mejores resultados. Debemos de reconocer que, en última instancia, este proceso no es más que una automatización del proceso de prueba por ensayo y error que realizáramos a mano, aunque tiene la ventaja de poder buscar mejor que nosotros entre todas los posibles agrupamientos. A continuación, veremos cómo podríamos usar realmente la programación genética para resolver el mismo problema (seguir una pared por la derecha).

## *Posible implementación con Programación Genética*

El objetivo es navegar en espacios abiertos hasta que se encuentre una pared y luego navegar sin alejarse demasiado de la pared. Hay que simular diferentes entornos para que las soluciones obtenidas sean más generales y no dependientes de un entorno particular. También hay que definir la función aptitud que evaluará la performance de cada individuo de la población en su comportamiento. Hay que traducir el comportamiento que buscamos a una función que califique a cada individuo, esto

implica definir una métrica del comportamiento. Por ejemplo, en [Dain 1998] se definió al mundo como una grilla y la función de aptitud toma en cuenta la cantidad de cuadraditos visitados cercanos a las paredes[16]. También hay que especificar las funciones con las que se construirá el programa que controle al robot. En el trabajo referido con anterioridad, se definen varias funciones: funciones para realizar operaciones secuenciales; funciones de chequeo (para las esquinas, para las distancias a las paredes, etcétera); y funciones para los movimientos del robot. Pero, entre estas últimas, además de las clásicas *ir_hacia_delante, doblar_a_la_izquierda, doblar_a_la_derecha*, usa otras como *Ponerse_paralelo_a_la_pared_más_cercana* que son simples de implementar en un simulador pero no sobre un robot real.

Generalmente, en los experimentos hay que reducir el número de individuos o de generaciones para reducir el tiempo de simulación de días a horas. Esto limita también la calidad de los comportamientos obtenidos. Otra desventaja de este método es que los programas obtenidos no son simples de entender (son árboles sintácticos) y, por ende, son más complicados para modificar posteriormente[17].

En definitiva, es posible implementar una solución pero es crítico contar con una función de aptitud adecuada que puede -al igual que nos ocurría con la función de aprendizaje por refuerzo- necesitar ser re-escrita para desalentar comportamientos de individuos que son aptos según nuestra función de aptitud pero no se desempeñan como deseamos. Además, tenemos que diseñar las funciones necesarias para construir la solución buscada (como los ladrillos de una casa). Dependiendo de las funciones que diseñemos, obtendremos mejores o peores desempeños en los robots y soluciones más simples o más complejas. Por esa razón hay que tener una idea aproximada de cómo se resuelve la tarea y no solamente qué queremos que el robot haga. Este es un detalle importante que diferencia la programación genética del aprendizaje por refuerzo. Ambos métodos tienen también semejanzas, la función de aprendizaje por refuerzo es conceptualmente similar a la función de aptitud. También restringimos la solución que puede hallar el robot, en la programación genética limitamos a través de las funciones que definimos nosotros mismos, y en el aprendizaje por refuerzo mediante los agrupamientos, las predisposiciones y la función de exploración.

### 3.2.2 Robot recolector

#### *Desarrollo*

Con la experiencia adquirida enfrentamos el desafío consistente en programar al robot para que pudiera ir hasta la "comida" y luego empujarla hasta llevarla a su "nido". Para eso, tenemos como datos: las coordenadas del robot, la de la comida (representada como una lata en el simulador) y las del nido. Está programado de manera que las coordenadas del nido son constantes pero, en caso de querer cambiar dicha posición, alcanza con modificar el valor y recompilar el archivo *sim.cpp*. El aprendizaje es

[16] la ecuación en realidad diferencia los cuadraditos del corredor de los cercanos a la pared; la distancia al centro de un corredor y la distancia recorrida.
[17] para que se entienda cuán difícil puede ser leer el código generado, se incluye en el apéndice el programa obtenido de uno de los individuos con mejor desempeño de los experimentos de [Dain 1998]

independiente de la posición del nido, es decir, que es posible cambiarla después del aprendizaje y usando los mismos valores de la función Q el robot deberá lograr su objetivo. La posición de la comida es conocida en el momento de la ejecución, pero es elegida aleatoriamente al comienzo de cada ejecución (se setea MOVE_OBSTACLES 1 en el *test.opt*). La posición inicial del robot también es elegida también aleatoriamente por la simulación. Con esto resulta que nuestra aplicación es robusta frente a cambios en la ubicación del nido, de la "comida" y posición del robot. Se asume que estamos en un mundo abierto, sin paredes ni otros robots ni obstáculos, sólo el robot y el objeto, y que el nido es un entorno de cierto tamaño alrededor de una posición dada.

**Definimos los estados de la siguiente forma:**

Agrupamos el conjunto de sensores de dos en dos:

- (sensor0+sensor1): *sensorIzquierdo*;
- (sensor2+sensor3): *sensorMedio*;
- (sensor4+sensor5): *sensorDerecho*;
- (sensor6+sensor7): *sensorTrasero*.

De esta manera tenemos por el momento cinco estados:

- *sin lectura;*
- lectura en el *sensorIzquierdo*;
- lectura en el *sensorMedio*;
- lectura en el *sensorDerecho*;
- lectura en el *sensorTrasero.*

A su vez, para cada uno de esos estados, el robot puede estar dirigido[18] hacia alguno de los cuatro cuadrantes mostrados en la figura 60.

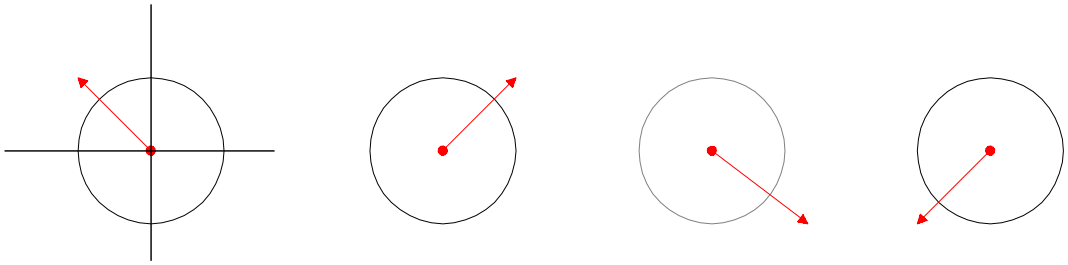


**Figura 60: cuatro orientaciones del robot (el círculo representa el cuerpo del robot visto desde arriba)**

Tenemos hasta el momento definidos veinte estados, pero las acciones que se deben tomar para resolver la tarea dependen también del cuadrante en el que se encuentre el robot con respecto del nido. Mostramos en la figura 61 los cuadrantes que estamos mencionando.

[18] El robot posee una propiedad denominada Dirección que puede ser consultada.

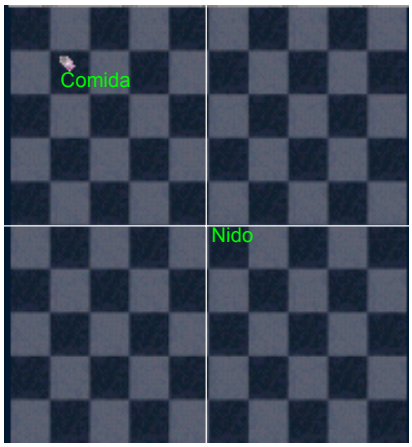



**Figura 61: cuatro cuadrantes según el nido (situado en este caso en el medio del mundo)**

Así que finalmente decidimos definir 80 estados (bastante más que en la tarea de seguir las paredes!!) considerando que era una buena aproximación de la realidad como para poder diseñar el aprendizaje.
Dejamos planteadas las mismas cuatro acciones de la tarea anterior: *girar a la izquierda*, *girar a la derecha*, *avanzar*, *retroceder*. Con esto tenemos una tabla para los valores de la función Q con 80x 4 = 320 entradas (filas).
Si el robot se aproxima demasiado a las paredes como para percibir una lectura en sus sensores confundiéndola con la lata ó si sale de los límites del mundo definido, terminamos la simulación escribiendo los valores de la función Q aprendidos en el archivo *qvalSal.txt* y la cantidad de veces que fue visitado cada par <estado,acción> en el archivo *nroQSal.txt*[19].
La función de recompensa consistía en: +1 si la distancia del objeto al nido era menor luego de haber tomado una acción dada; -1 si la distancia era mayor y 0 si era igual.

El resultado obtenido en los primeros experimentos no fue el esperado; observamos que si se encontraba en una posición favorable -entendiendo por favorable a una posición donde el objeto se encuentre entre el robot y el nido (figura 62)-, lo arrimaba al nido bastante bien, pero si quedaba en una posición desfavorable entre el objeto y el nido (figura 63), no llegaba a aprender que debía de rodear al objeto para luego, empujarlo. En algunos casos lograba rodear al objeto por azar, pero en otros quedaba estancado oscilando.

[19] Esto permite continuar el aprendizaje entre varias ejecuciones, para eso deben de renombrarse estos archivos a: *qval.txt* y *nroQ.txt* respectivamente, estos seran tomados como entrada en la siguiente corrida del programa.


**Figura 62: situación favorable**

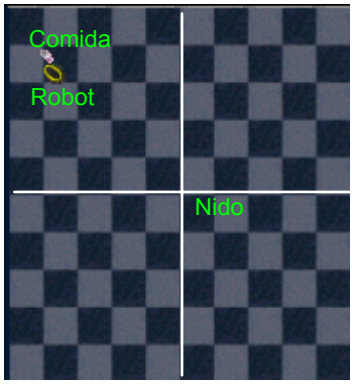



**Figura 63: situación desfavorable**

Igual obtuvimos esporádicamente buenos resultados mientras el robot aprendía, incluso algunos extraños como el que podemos observar en la figura 64.

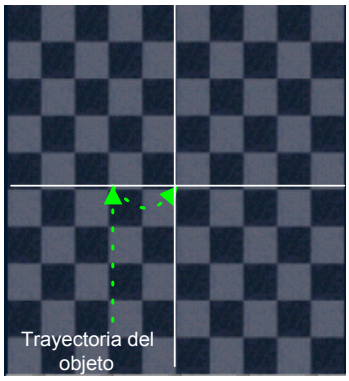



**Figura 64: solución extraña (aunque correcta)**

Esta solución extraña se debió a que premiábamos al robot cada vez que acercaba la comida al nido sin importar cuánto lo aproximara. Esto lleva a que sería posible cualquier trayectoria con tal que se vaya arrimando, incluso podría llegar al caso extremo (por lo largo del recorrido) de arrimarse moviéndose a lo largo de una espiral con centro en el nido (figura 65). Por esta razón modificamos la función de recompensa para que fuera proporcional a la distancia arrimada.

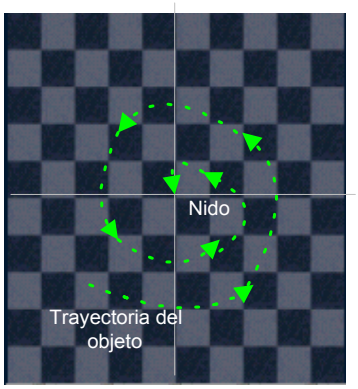



**Figura 65: trayectoria en espiral**

En el primer "borrador" de nuestra solución definimos un entorno alrededor de la "comida", colocamos inicialmente al robot en dicha área y controlamos que el mismo se moviera dentro de ella; si se salía del área, deshacíamos el movimiento. Vale aclarar que la tarea de deshacer no es aplicable en un robot real, pero estábamos aproximando una solución.

Al contemplar que era muy común que, en posiciones desfavorables, el robot quedara dentro del área definida oscilando sin empujar al objeto a causa de que cualquier acción es igualmente neutra, decidimos elaborar una segunda versión. Se nos ocurrió poner un contador de acciones que no daban premio y luego de cierta cantidad de acciones seguidas sin obtener premio, optábamos por aumentar el umbral de explotación. Teníamos la esperanza de que, al volver a explorar, el robot pudiera elegir una secuencia de acciones que lo llevaran a salir de esa oscilación y una vez obtenida una acción que le daba premio, volvíamos el contador a cero. En la práctica no observamos una mejora del comportamiento ya que, si bien logramos que no quedara oscilando, tampoco pudimos evitar que alejara bastante al objeto del nido debido a que lo habíamos hecho elegir una acción al azar hasta que tomara una que arrimara al objeto. Finalmente descartamos la idea del área y programamos un instinto para que el robot lograra encontrar el objeto si llegaba a perder contacto con el mismo.

En la tercera versión agregamos un instinto que se activaba cuando el robot no percibía al objeto. El mecanismo es el siguiente: si el robot no percibe al objeto, entonces se comporta según nuestro instinto hasta que vuelve a encontrarlo; y una vez hallado, el comportamiento pasa a ser el aprendido según el algoritmo de *q-learning*. Con esto logramos eliminar la necesidad de que el robot partiera de una posición inicial cercana al objeto; ahora teníamos un robot capaz de encontrar su "comida" conociendo sus coordenadas.

**Pseudocódigo del instinto para encontrar al objeto:** Este instinto hace que el robot busque y se aproxime al objeto hasta que lo percibe con sus sensores.

*Siendo conocida la dirección del robot (hacia donde está enfrentado), el instinto consiste en:*

*1º Si no hay lectura en los sensores (si todavía no encontró al objeto)*
*Si la diferencia entre la dirección del robot y la que debiera tener para estar enfrentado al objeto (dirección que une el centro del robot y el centro del objeto) es mayor a un cierto* $\delta$[20] *o menor a* $-\delta$ ( figura 66)*, entonces el robot gira a la derecha buscando enfrentarse al objeto*
*Si no* (figura 67)*: avanza.*
*2º Establezco el estado actual del robot.*
*3º Si se salió del mundo o está muy cerca de las paredes escribir valores de la función Q y terminar simulación.*

El código del instinto está en el apéndice C2.

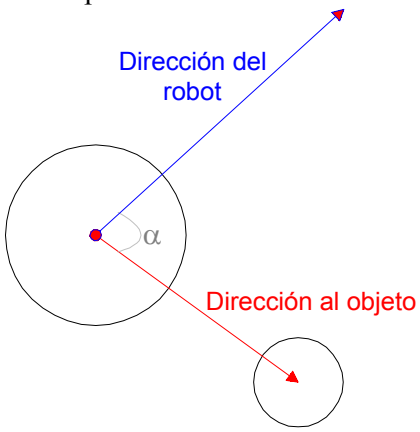



**Figura 66: caso donde la diferencia (α) es mayor a** δ.

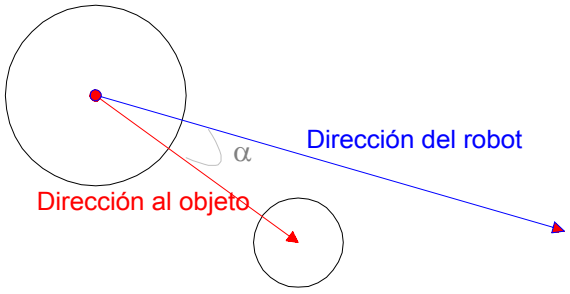



**Figura 67: caso donde la diferencia (α) es menor a** δ

Si bien con esto logramos eliminar el problema de que el robot se quedara girando y oscilando sin empujar al objeto, todavía nos ocurría que eventualmente lograba el

[20] En nuestros experimentos $\delta=30º$.

objetivo propuesto, pero si llegaba a una posición no favorable, seguíamos teniendo problemas para rodear. Debido a que agrupamos muy gruesamente los estados del mundo el robot no podía aprender por sí mismo que ciertos movimientos lo llevarían a rodear al objeto encontrándose en una mejor situación para empujarlo. Esto ocurría por la forma como habíamos definido los estados: el robot detectaba al objeto o no, pero no podía reconocer matices como *estar muy cerca* (donde seguramente lo empuje) o *estar cerca* (puede rodearlo). Entonces, cuando detectaba al objeto no sabía si estaba cerca o lejos y, si estaba en una posición desfavorable, algunas veces, *girar a la izquierda* podía ser bueno porque era un paso dentro de la maniobra *rodear al objeto* y otras veces era malo porque estaba alejándolo del nido. Podríamos haber enseñado a rodear con otra función de aprendizaje y luego combinar ambos comportamientos, pero cuando intentamos reusar el aprendizaje de seguir paredes -para que rodeara al objeto siguiendo su contorno- detectamos que el robot chocaba bastante al objeto, por lo que el hecho de *rodear* alejaba al objeto del nido perjudicando la tarea. De manera que optamos por otra alternativa más simple: decidimos programar por completo ese comportamiento como un instinto y no como un comportamiento aprendido.
Esto nos llevó a la cuarta versión, donde agregamos un instinto para que resolviera el problema de las posiciones no favorables como la que ilustramos en la figura 63.

**Pseudocódigo para resolver posiciones no favorables:**

*1º Mientras el robot esté muy cerca del objeto, se aleja para poder, posteriormente, maniobrar con comodidad. Esto lo hace porque, aunque haya perdido al objeto, puede ser que esté lo suficientemente cerca como para chocarlo mientras intenta posicionarse adecuadamente.*

*2º Si se separó lo necesario por su coordenada X, ahora debe corregir su posición en el eje 0Y hasta que el objeto quede colocado entre el robot y el nido si se considera solamente el eje 0Y. Luego, de ser necesario, debe corregir su posición en el eje 0X (ya sabemos que se alejó una distancia prudente por el eje X, pero la coordenada X del robot puede estar entre el objeto y el nido y por tanto necesitar corregirse).*
*Si no (si no se separó lo suficiente por sus coordenada X entonces se separó por las coordenada Y) hacemos que el robot corrija su posición en el eje 0X. Y luego, de ser necesario debe corregir también su posición en el eje 0Y.*

El código del pseudocódigo está en el apéndice C2.

**Pseudocódigo de cómo quedó el programa principal:**

*Declaración de variables*
*Inicialización de la simulación*
*Inicialización de los valores de la función Q*
*Determino el estado inicial*

*Por siempre*[21] *{*
    *Si el robot no percibe al objeto {*
        *Si está lejos del objeto ó está en posicion_favorable*[22]*{*
            *Se arrima con Instinto para encontrar al objeto*
        *}*
        *Si no { resolver posición no favorable }*
    *}*
    *Sino { Q-learning }*
*}*

De esta manera logramos que el robot cumpliera su objetivo y empujara al objeto hasta el nido. Pero llegamos a la conclusión de que lamentablemente el robot tuvo poco para aprender ya que una vez colocado en una posición favorable, casi cualquier movimiento que realice va a hacerlo chocar al objeto y, por ende, lo va a empujar hacia el nido.

Como no queríamos resolverle todos los problemas, hicimos una quinta versión donde el robot consideraba como posiciones favorables a aquellas donde el objeto estaba entre el robot y el nido según una de sus coordenadas en lugar de exigir que fuese según las dos. Con esto buscábamos que el robot aprendiera más movimientos al encontrarse en una situación no tan cómoda. Si bien aprendió algo más, fue bastante malo el comportamiento obtenido y muy pequeña la simplificación en nuestra programación por lo que volvimos a la versión anterior.

Teníamos programadas dos de las tres tareas involucradas en este problema. El *q-learning* nos ahorró programar el *empujar* pero hicimos a mano el *encontrar* y el *rodear*. Realizamos un intento más, donde el único instinto que dejamos fue el necesario para encontrar al objeto y pretendíamos que el robot aprendiera a rodear. Para eso, refinamos la definición de los estados; primeramente recordamos que para seguir paredes el sensor importante era el de la derecha y, en este caso, queríamos que el robot rodeara al objeto, por eso desagrupamos los sensores delanteros. Además, como era importante que el robot pudiera distinguir mejor tanto la cercanía como la lejanía, introdujimos un matiz en la detección del objeto, pero únicamente para los sensores delanteros, así que teníamos las posibilidades *cerca, lejos* o *sin_detectar*. De esta manera incrementamos el conjunto de estados a un total de 224 (contra los 80 anteriores): 6 sensores delanteros x 2 clase de lecturas + 1 sensor trasero con lectura + *sin lectura* en ningún sensor da un subtotal de 14 estados, para cada uno de esos estados hay una orientación que puede corresponder a cuatro cuadrantes y a su vez el robot puede estar posicionado en uno de cuatro cuadrantes según el nido (14x4x4=224).

[21] Dentro del bucle hay controles para que en caso que el robot arrime al objeto hasta el nido ó tanto el robot como el objeto se salgan del mundo, se guarden los valores aprendidos y termine la simulación.
[22] Posición favorable si el robot y el objeto están en el mismo cuadrante según el nido y ambas coordenadas del objeto están más cercanas al nido que las del robot (ver figura 62).

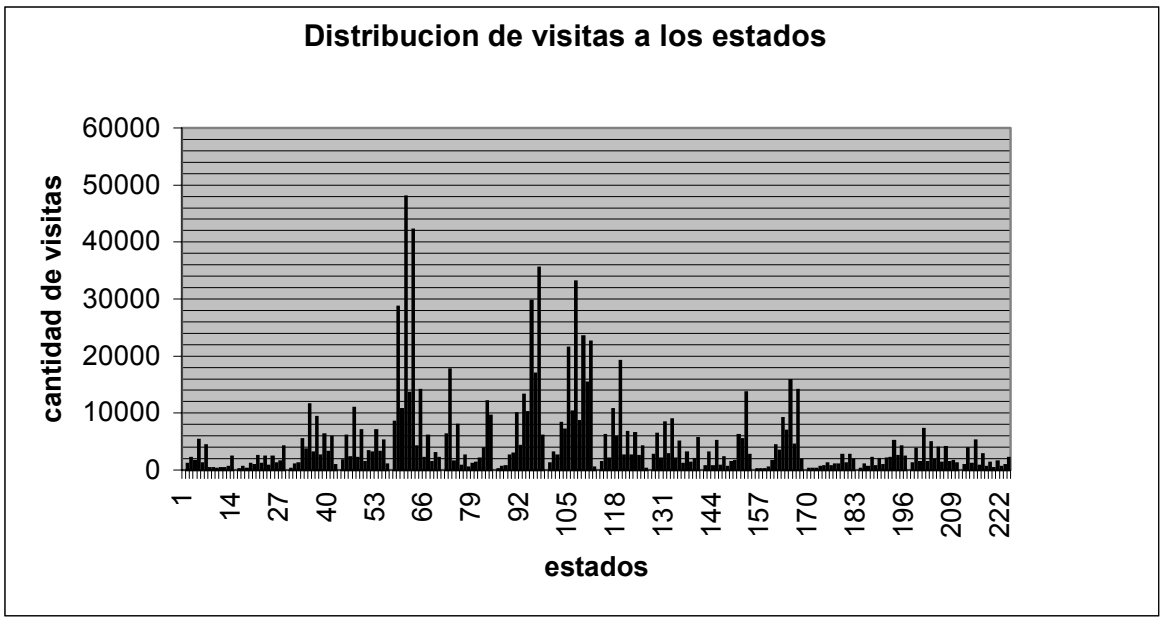



**Figura 68: distribución de visitas por cada estado luego de 1 millón de iteraciones**

Luego de un millón de iteraciones observamos una distribución en la cantidad de visitas como se observa en la figura 68. Como se puede constatar, el cuadrante 2 (estados 56 al 111) se ha visitado considerablemente más que el resto. Tal concentración no se debe a una peculiaridad del cuadrante 2 sino a que, seguramente por casualidad, fue más visitado en algún momento del aprendizaje y se ajustaron más sus valores para la función Q. Además en varias ocasiones, cuando comienza la simulación con el objeto en otro cuadrante, se puede llegar al cuadrante 2 y quedarse moviendo con oscilaciones que incrementan aún más el número de visitas de este cuadrante. El aprendizaje obtenido, luego de un millón de iteraciones, no es del todo correcto, razón por la que, en varias ocasiones, el robot queda oscilando entre dos estado. Esto quizás se deba a que el robot puede rodear el objeto por la izquierda o por la derecha o porque el agrupamiento continúa siendo demasiado grueso. La gráfica de la figura 68 presenta el eje OY dividido cada 2000 visitas; nuestro umbral de explotación disminuye desde un 100% hasta un 5% a lo largo de 4000 visitas a cada estado, por lo que se puede observar que para varios estados -sobre todo en los cuadrantes 1 y 4- todavía el umbral de explotación viene disminuyendo (no terminamos el aprendizaje). Esto podría hacer pensar que el comportamiento en el cuadrante 2 debiera ser muy superior al del cuadrante 1. Sin embargo, como se observa en la tabla 1, pese a la enorme diferencia en la cantidad de visitas, el comportamiento obtenido no es tanto mejor en cuanto a sus resultados promedio (un 36% menos de iteraciones). Esta tabla muestra 10 ejecuciones desde los mismos valores de la función Q aprendidos luego de un millón de iteraciones y un comportamiento promedio (para el mismo no consideramos los valores extremos máximo y mínimo que distorsionaban el promedio). Elegimos una posición arbitraria en el cuadrante superior derecho (x= 600; y= 300) y su posición espejo[23] en el cuadrante superior izquierdo (x= 400; y= 300). Como se puede observar, hay una gran varianza en los resultados y hay ejecuciones donde el cuadrante izquierdo (cuadrante1) presenta un mejor comportamiento que otras ejecuciones del cuadrante derecho (cuadrante 2), no obstante en promedio el cuadrante derecho se resuelve mejor.

[23] Espejo respecto al nido que se encontraba en la posición (500,500).

| Comparación | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cuadrante 2 | ***Iteraciones*** | 1044 | 3497 | 3764 | 1650 | 2818 | 4383 | 12499 | 4963 | 359 | 4309 | 3303 |
| | ***Premios*** | 36 | 35 | 43 | 45 | 39 | 38 | 64 | 36 | 33 | 38 | 38 |
| | ***Castigos*** | 0 | 0 | 0 | 7 | 0 | 0 | 16 | 0 | 0 | 0 | 2 |
| | ***Recompensa Promedio*** | 37 | 38 | 30 | 25 | 33 | 35 | 15 | 37 | 41 | 35 | 33 |
| Cuadrante 1 | ***Iteraciones*** | 1828 | 23102 | 6172 | 1064 | 1692 | 5051 | 1179 | 7804 | 10177 | 2019 | 4490 |
| | ***Premios*** | 47 | 164 | 61 | 51 | 47 | 61 | 46 | 66 | 56 | 52 | 55 |
| | ***Castigos*** | 8 | 111 | 3 | 1 | 7 | 10 | 6 | 5 | 3 | 8 | 6 |
| | ***Recompensa Promedio*** | 23 | 2 | 19 | 25 | 24 | 17 | 24 | 17 | 21 | 21 | 18 |

**Tabla 1: comparación de performance entre cuadrantes 1 y 2**

## *Discusión*

### Estados, Acciones e Iteraciones

#estados= valores_sensores x valor_posición x valor_orientación
#acciones= valor_rueda_izquierda x valor_rueda_derecha

#estados= $10^8$ x (1000 x 1000) x 360 = O($10^{16}$); en esta discretización más fina del conjunto de estados hay también un agrupamiento subyacente.

#acciones= 10 x 10= 100; estos 10 valores posibles para cada motor es en los hechos también un agrupamiento.

Nosotros lo simplificamos a: #estados= 80 ó 224 (según la versión); y #acciones=4.
Es entendible que nuestro modelo del mundo sea bastante más simple que si aprovecháramos toda la capacidad del robot.

Programar al robot recolector tiene algunas características inconvenientes:

- El espacio de estados es muy grande y hay que buscar métodos para generalizar y propagar la recompensa.
- La idea de la recompensa es difícil para las 3 tareas a la vez.
- El robot aprende en el límite con el objeto (en nuestros experimentos con el agrupamiento más fino, que tiene 224 estados, solamente en un 1 ó 2% de las iteraciones obtenía una recompensa no nula)

Nos llevó unas 600.000 iteraciones de aprendizaje que lograra remontar la recompensa promedio a un valor mayor a cero. Consideramos que -incluso luego de un millón de iteraciones- lo aprendido puede mejorarse bastante. Aunque luego de ver la tabla 1, parece ser que por más aumentemos la cantidad de visitas a cada estado, el aprendizaje mejora muy lentamente. Tal vez estamos relativamente cerca del comportamiento óptimo que resulta posible obtener según nuestro diseño de estados y acciones. En esta instancia es posible observar cierta oscilación entre estados; esto se debe principalmente

a que todavía no terminó de aprender una política. Dada la observación de que dicha oscilación se da, sobre todo, en las posiciones desfavorables, podríamos argumentar también que esto se debe a que el robot no tiene una preferencia al momento de rodear al objeto por la derecha o por la izquierda siendo posible que estuviera 'indeciso'. Otro asunto a tener en cuenta es que podría ocurrir que una acción en un estado particular diera mucha recompensa, pero que fuese difícil llegar a él. Por ejemplo, porque en la cadena estado-acción que habría que ir recorriendo para alcanzarlo, se van reduciendo las exploraciones, y quizás cuando finalmente alcancemos ese valioso estado ya no se propague mucho su valor a causa de que $\beta$ sea demasiado pequeño.

Con respecto a la condición de parada del aprendizaje si sale del mundo, como nuestro algoritmo está diseñado para un mundo infinito y abierto, carente de paredes, tuvimos que controlar que se mantuviera dentro de ciertos límites manejables. En la definición de los estados no llegamos a manejar una noción de proximidad al borde del mundo y salirse del mismo no es fallar ni es castigado. De hecho, puede ser que esté efectivamente arrimando al objeto, por ejemplo, a través de una trayectoria en espiral y que se salga del mundo. Si tuviéramos un mundo infinito, podríamos, en teoría, correr nuestro algoritmo por siempre y, luego de mover mucho al objeto mientras aprende (alejándolo y acercándolo), veríamos que el robot finalmente aprende y en algún momento llega al origen con él.

## Comparación entre las dos soluciones

Luego de sucesivas versiones tenemos dos soluciones satisfactorias. Una donde tenemos un agrupamiento menos refinado pero dividimos el problema en tres tareas: *encontrar* (programada a mano), *rodear* (a mano) y *empujar* (aprendida con *q-learning*). Y la otra con un agrupamiento más fino donde dividimos el problema en dos tareas: *encontrar* (a mano) y *rodear-empujar* (aprendida).

Para comparar las soluciones asumimos que, en sectores simétricos del escenario respecto del nido, el robot debería haber aprendido soluciones simétricas. Luego dividimos uno de los cuadrantes en 9 partes iguales (como se muestra en la figura 69). Y para no centrar toda la comparación en un único cuadrante, respetando la simetría, repartimos las 9 particiones entre los cuadrantes (como se muestra en la figura 70). Elegimos una posición en cada partición, efectuamos diez ejecuciones con cada algoritmo y promediamos. (Ver el apéndice B donde publicamos todos los valores de los experimentos que realizamos para promediar y elaborar la tabla 2. Allí se pueden observar los valores máximos y mínimos para cada caso, y la varianza de los mismos).

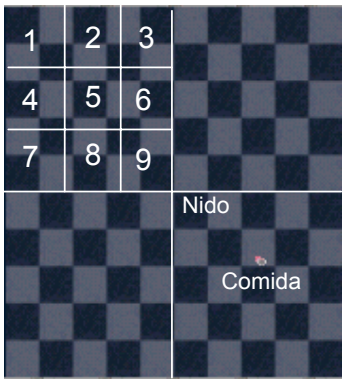



**Figura 69: partición del cuadrante 1**


**Figura 70: distribución de las 9 particiones**

En los resultados que mostramos en la tabla 2 se muestra claramente la superioridad de la solución donde usamos dos instintos. Efectuamos solamente 10.000 iteraciones de aprendizaje (en comparación al 1.000.000 de la otra aplicación), y sin embargo, en la tabla se observa que obtuvimos resultados similares o mejores. Para contar las iteraciones sumamos 1 por cada acción tomada; en el caso de la aplicación con dos instintos, también contamos las iteraciones correspondientes al instinto *dar_la_vuelta* para que la comparación sea posible (sino estaríamos comparando la tarea de *empujar* en una, contra la tarea de *rodear-empujar* en la otra).

La velocidad para aproximar el objeto al nido es mayor en la aplicación con dos instintos. Es coherente que aprenda más rápido ya que, como programamos a mano el *rodear* debe aprender menos (sólo a *empujar*). En la otra aplicación, sin embargo, tiene que darse cuenta de la necesidad de *rodear* por sí mismo usando la misma función de recompensa -que no fue diseñada para enseñarle a *rodear,* sino a *empujar*-.

| ***Comparación*** | | **Promedio (150,150)** | **Promedio (600,300)** | **Promedio (600,900)** | **Promedio (250, 900)** | **Promedio (250,250)** | **Promedio (750,410)** | **Promedio (910,600)** | **Promedio (120,750)** | **Promedio (400,400)** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Agrupamiento fino** | ***Iteraciones*** | 4443 | 3303 | 1054 | 6240 | 8253 | 3599 | 2625 | 3813 | 3965 |
| | ***Premios*** | 114 | 38 | 90 | 140 | 96 | 56 | 76 | 136 | 23 |
| | ***Castigos*** | 7 | 2 | 3 | 25 | 15 | 2 | 1 | 19 | 5 |
| | ***Recompensa Promedio*** | 29.8 | 33 | 36.62 | 25 | 25.67 | 34.39 | 44.75 | 23.43 | 25.54 |
| **Dos instintos** | ***Iteraciones*** | 856 | 3364 | 1542 | 526 | 2202 | 416 | 1814 | 603 | 180 |
| | ***Premios*** | 93 | 50 | 77 | 90 | 74 | 40 | 81 | 135 | 14 |
| | ***Castigos*** | 1 | 5 | 0 | 0 | 4 | 0 | 2 | 0 | 0 |
| | ***Recompensa Promedio*** | 42 | 27 | 41.94 | 42.77 | 33.5 | 44.04 | 39.59 | 40.54 | 40 |

**Tabla 2: comparación entre las dos soluciones**

En la tabla 2 podemos ver que en la posición (400,400), la solución obtenida con un agrupamiento más fino es, realmente, bastante lenta si tenemos en cuenta la cercanía de

esta posición al nido (500, 500). Esto se debe a que el cuadrante superior izquierdo no está lo suficientemente explorado; además, esta posición está muy cerca de las fronteras entre los cuadrantes. Todo ello ocasiona situaciones en donde el robot está en un cuadrante y el objeto en otro, las cuales no fueron consideradas especialmente en el modelado y entorpecen un poco el aprendizaje. Asimismo observamos que el comportamiento aprendido hasta el momento consiste en empujar hacia el cuadrante superior derecho y allí también se encuentra con una situación difícil pues debe rodear al objeto. Observamos que para los casos (150,150) y (250,250), que también caen en el cuadrante superior izquierdo (en 1 y 5 respectivamente en la figura 70), la cantidad de iteraciones es incoherente para ambos algoritmos ya que parece demorar en promedio menos para empujar desde la posición más alejada del nido (150,150) que desde la otra más cercana (250,250). Esto se debe a que, en realidad, los resultados están viciados porque muchos de los experimentos en la posición (150,150), al estar tan cerca del borde, terminaron fuera del mundo útil, lo cual implica que el promedio solamente toma en cuenta los mejores casos.

Ahora que tenemos una mejor idea del comportamiento global de los dos algoritmos, veremos un poco más a fondo como evoluciona la distancia al nido según ambos algoritmos para una de las posiciones (600,300).

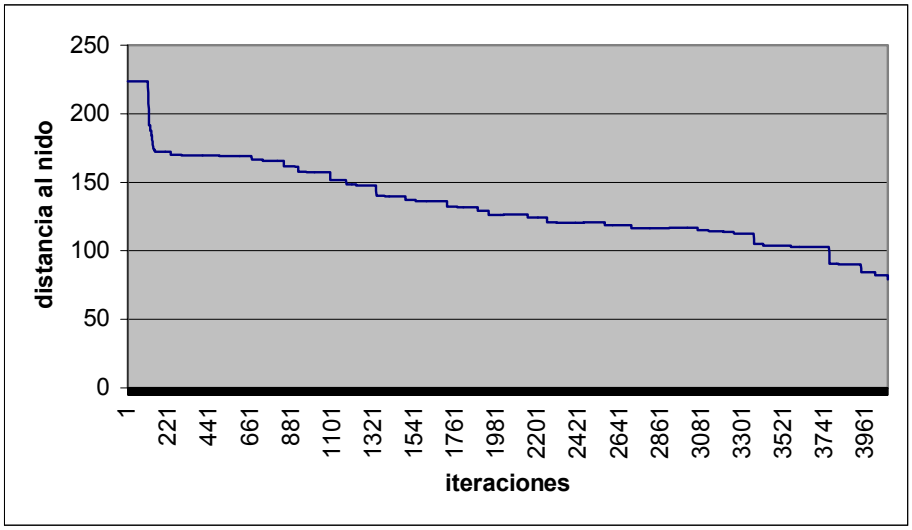



**Figura 71: gráfica de la distancia al nido según el avance de las iteraciones (dos instintos)**

En la figura 71 se puede observar la evolución de la distancia al nido durante una ejecución del algoritmo con dos instintos desde la posición inicial (600,300). Se ve cómo a lo largo de poco más de 4.000 iteraciones va disminuyendo la distancia al nido hasta que llega a ser menor a 80 (es una de las condiciones de parada del algoritmo, significa que logró su objetivo). Las "mesetas" -en la gráfica- corresponden a acciones que no afectan la posición del objeto, principalmente se deben al instinto *dar_la_vuelta* programado por nosotros.

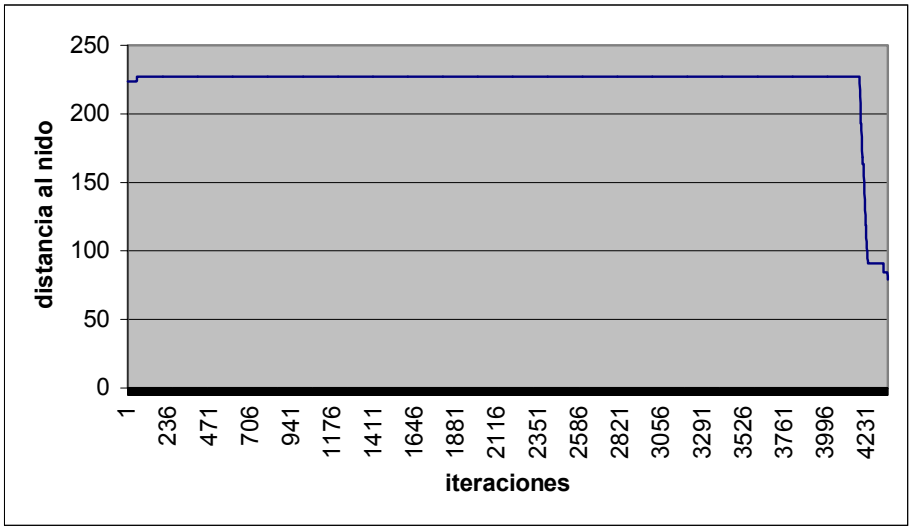



**Figura 72: gráfica de la distancia al nido según el avance de las iteraciones (agrupamiento fino)**

En la figura 72 se puede observar una ejecución para el mismo caso que en la figura 71, pero siendo solucionado por el algoritmo con un solo instinto y un agrupamiento más fino. Se aprecia que lo que más tiempo le llevó al robot fue rodear al objeto y luego, afortunadamente, no tuvo mayores inconvenientes. Cada meseta representa una serie de acciones que no arriman ni alejan al objeto.

El algoritmo con dos instintos y un agrupamiento más grueso aprendió en 10.000 iteraciones lo que el otro, con un solo instinto y agrupamiento más fino, aprendió en 1.000.000 de iteraciones (basados en la Tabla 1, es probable que en realidad sea similar a lo aprendido en 600.000 iteraciones). La versión con dos instintos presenta, en general, una mejor performance variando desde sectores donde se comporta casi igual a la versión con un agrupamiento más fino hasta otros donde es 22 veces más rápida. Así que, en términos de performance, es mejor dividir la tarea en subtareas con sus respectivos agrupamientos. Pero si se dispone del tiempo suficiente, es posible efectuar un agrupamiento más fino y un aprendizaje más largo obteniendo asimismo resultados bastante buenos. Igualmente confiamos más en el algoritmo con dos instintos ya que sabemos que siempre podrá *rodear* bien y la tarea *empujar* no resulta tan difícil.

## Trabajo futuro

- Incluir otros robots;
- Incluir otros objetos-comida;
- Incluir paredes;
- Incluir otros nidos;
- Incluir una mejor técnica para manejar generalización entre estados;
- Probar sobre un robot real.

La mayoría de estas extensiones buscan darle mayor realismo a la tarea del recolector y tienen, principalmente, la misma dificultad: diferenciar la percepción de un objeto-comida de la de otros robots o paredes del entorno. Sería posible adaptar nuestra aplicación para la presencia de varios robots, pero en este caso, nuestros instintos

fallarían al percibir a otro robot ya que no contemplan esta posibilidad. Lo mismo ocurriría con el *q-learning* ya que los robots se provocarían lecturas entre ellos confundiendo al otro con el objeto a empujar y esto entorpecería el aprendizaje. En una aplicación real no resulta fácil diferenciar si está percibiendo a un robot o al objeto que debe empujar. En el simulador sí se pueden usar funciones para detectar si el robot está recibiendo lecturas a causa de la presencia de otro robot, razón por la cual -en el simulador- no sería un problema grave.

También cabría mejorar el código -para hacerlo mantenible y más fácil de extender en el futuro- definiendo al ESTADO, ACCION y QVALUES[24] como objetos (aprovechando que estamos programando en C++). Y se podría, opcionalmente, definir el objeto INSTINTO. Otra mejora sería incluir los parámetros (coordenadas del nido y el largo del mundo, entre otras) en el archivo *test.opt* en lugar de definirlas como constantes, lo cual evitaría tener que re-compilar el código si queremos cambiar estos valores.

Observando la tabla 2 en la posición (400,400), se podría pensar que, tal vez, los casos donde el robot está en un cuadrante y el objeto en otro interfieran más de lo previsto en el aprendizaje. Por ello podría aumentarse el espacio de estados para incluir estos casos y ver si mejoran los resultados.

## Trabajo relacionado

En [Mahadevan and Connell 1991] aparece el problema de empujar una caja usando aprendizaje por refuerzo. Lo resuelve dividiendo la tarea en 3: encontrar la caja; empujarla; y desatascarse (resolver situaciones donde el robot está atascado). Aprende cada tarea por separado (con diferentes funciones de recompensa para cada tarea). Este problema es distinto al resuelto por nosotros ya que el robot solamente debe empujar hasta que la caja llegue a una pared, además debe diferenciar las cajas de las paredes (cuando detecta que tiene algo enfrente lo intenta empujar y si lo logra entonces es una caja!). Comparan la performance frente a un intento de resolver el mismo problema pero en forma monolítica (sin dividir en tareas) llegando a la misma conclusión que nosotros: con la división obtienen una performance mejor (siete veces mejor); mantienen un umbral de exploración/explotación del 10% afirmando que, cuando el robot aprende una buena política, se puede recuperar de las eventuales desviaciones ocasionadas por una mala decisión al explorar; usan el comportamiento aprendido luego de 2000 iteraciones ya que en las pruebas que realizaron con un aprendizaje más largo no observaron una gran mejora (nuevamente esto no significa que si lo hubieran corrido durante un millón de iteraciones no hubiera mejorado, es solo su intuición personal; hay mucha heurística en esta área). También manejan otras acciones: *avanzar*, *girar_izquierda*, *girar_derecha*, *girar_mucho_izquierda* y *girar_mucho_derecha*. Para enfrentar el problema del tamaño del espacio de estados recurren a técnicas tales como: distancia de hamming con pesos; agrupamiento estadístico. También para ellos el comportamiento aprendido es algo decepcionante ya que el robot no aprende realmente a diferenciar las cajas de las paredes ni logra apilar las cajas contra las esquinas (que parece ser que era lo que realmente buscaban que ocurriera).

[24] Se refiere a valores de la función Q

# 4. Conclusiones Finales

Como vimos, hay varios enfoques para investigar en robótica; la fuente de inspiración por excelencia para diseñar técnicas y soluciones para los problemas de inteligencia artificial es el reino animal. Esto es debido a que allí encontramos los ejemplos más exitosos de comportamiento inteligente simple que conocemos. Todavía no contamos con las técnicas necesarias para elaborar comportamientos complejos, pero se tiene la esperanza de que puedan surgir de cierta interacción o combinación de comportamientos más simples. Hay motivaciones para desarrollar controladores de robots simples y eficientes con un enfoque *bottom-up* y otros *top-down*.

Al momento de diseñar una aplicación es necesario manejar cada vez más una mayor gama de herramientas, tales como redes neuronales con sus variantes, algoritmos genéticos, programación genética y aprendizaje por refuerzo, entre otras. Es posible que también sea preciso poseer un mayor conocimiento sobre disciplinas externas al área de computación, más humanistas. Esto parece inducir a la necesidad de trabajar en equipos multidisciplinarios con expertos de varias áreas diferentes. Teniendo en cuenta que la casi totalidad de los mecanismos utilizados actualmente para programar robots están basados en observaciones de zoólogos, sería aconsejable –aunque no indispensable- contar con una mayor formación en esa área.
Diseñar una aplicación para un robot no resulta ser una tarea simple de llevar a cabo todavía y no contamos con métodos eficaces ni eficientes para realizarla. Los métodos actuales exigen muchas características especiales para asegurarnos la obtención de buenos resultados e imponen restricciones que no son aplicables a la mayoría de los casos. Hay todavía tareas triviales para los humanos que resultan complejas de incorporar a los robots, como por ejemplo la visión, el lenguaje, y la abstracción, entre otras.

Las universidades se esfuerzan en hallar variantes a las técnicas para superar algunas limitantes de las mismas ó en adaptarlas para que funcionen especialmente bien en algún caso muy particular. Consideramos que es un error que, actualmente, se simplifiquen tanto las tareas o el modelado del entorno para lograr que los robots aprendan en horas en lugar de días ya que los animales y las personas aprendemos a lo largo de años. Quizás debiéramos ejecutar los programas durante meses para obtener soluciones mejores a las alcanzadas hasta el momento.

Una mejora en la calidad de la lectura en los sensores y un desarrollo de actuadores más versátiles aumentaría la potencialidad de los robots. Asimismo, una mayor potencia del hardware permitiría una mayor capacidad de cálculo para nuestros algoritmos, aunque quizás no se precise una mayor cantidad de cálculo sino una mejor calidad del mismo. Sin embargo, parece resultar indispensable poder tratar con un volumen de información cada vez mayor.

Pese a las carencias detectadas, estamos en un momento de particular auge de la robótica ya que se han logrado simular algunos comportamientos básicos con gran éxito; debido a estas carencias es que creemos que debiéramos investigar ya que hay muchas respuestas por hallar y oportunidades para crear nosotros mismos una solución a algunos de los cientos de problemas que restan por resolver. Igualmente es probable que nos falte todavía algún componente esencial que nos permita emular mejor los comportamientos que denominamos inteligentes.

Ha aumentado la cantidad de talleres y anuncios en las noticias, Internet, televisión y medios de difusión en general sobre avances y conferencias acerca de la Inteligencia Artificial, Redes Neuronales y Algoritmos Genéticos. También ha aumentado el número de competiciones para robots como Robot Soccer y Robot Wars entre otras. En el capítulo 2 se mencionó el interés de amateurs y profesionales con proyectos de diversa índole. Es que en cierto modo el hecho de programar aplicaciones con robots nos brinda la sensación de ser pioneros; algo que se está perdiendo en otras áreas de la computación hoy en día más típicas y con un mayor desarrollo. Esta tarea también tiene la peligrosa cualidad de hacernos sentir creadores de vida y jugar a ser dioses.

En las aplicaciones que fuimos efectuando pudimos constatar la complejidad de resolver tareas que son muy simples para nosotros los seres humanos, incluso, aquellas que, nunca nos ponemos a pensar cómo es que las realizamos-. Lo mismo les ocurre a los que intentan desarrollar robots que simplemente caminen como humanos. Dicha tarea llevó varios años de desarrollo y sólo recientemente se ha logrado emular con cierta performance aceptable. Otro aspecto difícil para desarrollar aplicaciones es tener que pensar en términos de sensores del robot y estados del mundo; estamos muy acostumbrados a nuestros sentidos y a sintetizar sus señales de forma instintiva. También observamos que hay que elegir entre un aprendizaje muy lento o resolver algunas situaciones programando a mano y dejar que se aprenda el resto.

En el aprendizaje por refuerzo, las funciones del mismo deben elegirse con cuidado; en el caso de la tarea de seguir paredes, optamos por definir la función según la transición entre estados; en cambio, en la tarea del robot recolector resultó más simple definirla basándonos en los resultados esperados (cuando el objeto estaba más cerca del "nido" premiábamos y si estaba más lejos castigábamos). Además, en la tarea de seguir paredes corroboramos que hay que tener cuidado con ciertas secuencias de acciones, como en el caso en el cual el robot se arrimaba y alejaba oscilando y recibiendo en total una recompensa de +99 (+100 + -1). Hay que impedir que el robot quede en un ciclo -entre estados- no deseado. Esto puede lograrse evitando premiar exageradamente ó, como se propone en [Touzet 1997], prohibiendo dichas secuencias una vez detectadas.
Seguidamente vimos que un agrupamiento inadecuado puede hacer que el robot nunca aprenda el comportamiento buscado; es necesario tener cuidado y un cierto grado de comprensión sobre cómo interactúa el robot con su entorno mientras lleva a cabo la tarea si se quiere agrupar. En la aplicación para seguir paredes intentamos agrupar de a dos sensores buscando darle robustez a la solución, pero resultaba imposible que el robot convergiera a la solución si no liberábamos el sensor de más a la derecha. Al aumentar la complejidad de la tarea, aumenta el número de estados (pasamos de 5 ó 9 a 80 y 224); resulta entonces importante poder contar con una forma eficiente de manejar operaciones con tantos estados y acciones. Esto también influyó al momento del aprendizaje: en la primer tarea alcanzó con unas miles de iteraciones para que se comportara bien; en cambio, para la segunda nos llevó 10.000 iteraciones para el caso donde programamos a mano dos instintos y cerca de 1.000.000 de iteraciones cuando hicimos que aprendiera a rodear. Lamentablemente no existe una cota que nos ayude a saber a priori cuántas iteraciones serán necesarias para desarrollar el comportamiento buscado. Esto hace que no sea posible saber cuánto tiempo vamos a tener que estar entrenando a nuestro robot, incluso, luego de muchos días de entrenamiento no podemos afirmar si la demora se debe a una lenta convergencia a la solución ó a que la función diseñada nunca llevará al comportamiento buscado (como cuando nos

percatamos, finalmente, que con un agrupamiento tan grueso el robot no iba a aprender nunca a rodear al objeto).
Al final del entrenamiento en la tarea del robot recolector, el comportamiento obtenido nos resulta bastante pobre. Si bien es correcto, en cuanto a que logra arrimar al objeto, es demasiado lento para empujar. Con lo aprendido luego de 1.000.000 de iteraciones y si, como se menciona en otros papers, en un robot real 2000 iteraciones corresponden aproximadamente a 2 horas, entonces estamos frente a un robot que puede demorar fácilmente 1 hora para empujar el objeto hasta cerca de 16 centímetros (un entorno) del nido desde una distancia inicial de 45 centímetros[25]. En la mayoría de los papers sobre aprendizaje por refuerzo existe una necesidad de poder lograr rápidamente el comportamiento esperado pero siempre se paga algún precio que impacta en la solución obtenida; quizás haya que tener más paciencia y apuntar a sistemas que aprendan durante días o semanas en lugar de horas. Otra posibilidad es dividir en tareas más simples, pero, como ya mencionamos, le estamos quitando expresividad al aprendizaje del robot y si hubiera varias maneras de dividir una tarea, tampoco es trivial hallar la mejor descomposición. Igualmente tenemos mucha confianza en la técnica de aprendizaje por refuerzo combinándola con otras que aporten velocidad al aprendizaje.

## Nota Final

Pensamos que en los próximos años veremos grandes avances en la robótica y no será extraño contar con un robot que reconozca caras y voces aún cuando falte mucho para que puedan mantener una conversación. Inicialmente desempeñaran algunas tareas repetitivas y rutinarias y, a medida que mejoren nuestros diseños, podremos tener robots-sirvientes para que limpien la casa, nos cocinen, nos manejen el auto y para todas aquellas tareas que podamos modelar. No hay que olvidar que se está intentando imitar la inteligencia, pero faltan aspectos tales como la emoción, el sentimiento, el miedo a perder y la ilusión. Las máquinas por el momento -y seguramente por los próximos 30 años- no gozan, no esperan, no ganan y no pierden. Por eso son perfectas, no son, en definitiva, tan complejas como nosotros. Podrá avanzar su aplicación en aquellas tareas que requieran únicamente de inteligencia, que son muchas, pero no son todas. Interesantemente este avance provocará un gran desempleo, excepto en aquellos lugares donde sea más importante manejar las emociones o la intuición, eso nos obligará a desarrollar mejor esas características ya que para trabajar se evaluará la inteligencia emocional en lugar del coeficiente intelectual. No deja de ser una paradoja: tanto razonamiento nos conducirá a prestarle más atención al mundo de las emociones y sensaciones. No podemos vaticinar ciertamente si será el fin del trabajo como lo conocemos, ni si todos nos dedicaremos de lleno a trabajos con énfasis en los sentimientos, como por ejemplo el arte o la comunicación. En lo personal, apostamos por que sea así; el próximo paso evolutivo del hombre será aprender más sobre sus emociones, conocerlas, manejarlas y desarrollarlas. Tenemos la esperanza de que llegaremos a ser mejores seres humanos, habremos calmado nuestra ansiedad de comprensión sobre el mundo físico y en ese momento entenderemos que desde el comienzo lo que siempre quisimos fue comprendernos a nosotros mismos.

[25] Para este cálculo usamos el valor de la posición (600,300), al robot le llevó aproximadamente 3300 iteraciones. El radio del robot es 27.5 en el simulador y 5.5 en la realidad por lo que hay una escala 1:5.

Consideramos y esperamos que esta tesis pueda servir como documento base y punto de partida para una mayor investigación en el área de la robótica en el Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República.

# Bibliografía

[Aljibury and Arroyo 1999] Hal Aljibury and A. Antonio Arroyo, *Creating Q-table parameters using Genetic Algorithms*, 1999 Florida conference on Recent Advances in Robotics, University of Florida, 1999.

[Brooks 1986a] Rodney A. Brooks, *A Robust Layered Control System for a Mobile Robot*, IEEE Journal of Robotics and Automation, RA-2, 14-23, April, 1986.

[Brooks 1986b] Rodney A. Brooks, *Achieving Artificial Intelligence Through Building Robots*, A.I. Memo 899, MIT Artificial Intelligence Laboratory, May, 1986.

[Brooks 1990] Rodney A. Brooks, *Elephants Don't Play Chess*, 1990.

[Brooks 1991a] Rodney A. Brooks, *Intelligence Without Representation, Artificial Intelligence*, 47, 139-159, 1991.

[Brooks 1991b] Rodney A. Brooks, *The Role of Learning in Autonomous Robots*, MIT Artificial Intelligence Laboratory, 1991.

[Budenske and Gini 1992] John Budenske and Maria Gini, *Achieving Goals Through Interaction with Sensors and Actuators*, IEEE/RSJ International Conference on Intelligent Robotics and Systems, 1992.

[Dain 1998] Robert A. Dain, *Developing Mobile Robot Wall-Following Algorithms*, HTR Labs, Applied Intelligence 1998; 8(5):33-41.

[Hailu and Sommer 1999] G. Hailu and G. Sommer, *On Amount and Quality of Bias in Reinforcement Learning*, IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC'99)- pages 1491-1495, Tokyo, Japan.

[Haykin 1994] Simon Haykin, "Neural Networks. A comprehensive foundation", Macmililan College Publishing Company, 1994.

[Lau, Tan, Erwin and Petrovic 1999] Kin W. Lau, Heng Kiat Tan, Bejamin T. Erwin and Pavel Petrovic, *Creative Learning in School with LEGO Programmable Robotics Products*, Frontiers in Education Conference, 1999.

[Laurent and Piat 2001] Guillaume Laurent and Emmanuel Piat, *Parallel Q-Learning for a block-pushing problem*, Laboratoire d'Automatique de Besançon, Francia, 2001.

[Lee and Zhang 2000] Kwang-Ju Lee and Byoung-Tak Zhang, *Learning Robot Behaviors by Evolving Genetic Programs*, IEEE, Seoul National University, Korea, 2000.

[Kaelbling, Littman and Moore 1996] Leslie Pack Kaelbling, Michael L. Littman and Andrew W. Moore, *Reinforcement Learning: A Survey*, 1995.

[Mahadevan and Connell 1991] Sridhar Mahadevan and Jonathan Connell, *Automatic Programming of Behavior-based Robots using Reinforcement Learning*, IBM T.J. Watson Research Center, 1991.

[Nehmzow and Mitchell 1995] Ulrich Nehmzow and Tom Mitchell, *The Prospective Student's Introduction to the Robot Learning Problem*, Department of Computer Science, University of Manchester, Technical Report Series UMCS-95-12-6.

[Robinson 2002] Albert Robinson, *Final Project: Reinforcement Learning*, 2002.

[Santos 1999] Juan Miguel Santos, *Contribution to the study and the design of reinforcement functions*, PhD tésis, 1999.

[Santos and Touzet 1999] Juan Miguel Santos and Claude Touzet, *Dynamic Update of the Reinforcement Function during Learning*, *Connection Science*, Vol. 11, 1999.

[Sathiya and Ravindran 1995] S. Sathiya Keerthi and B. Ravindran, *A Tutorial Survey of Reinforcement Learning*, Department of Computer Science and Automation, Indian Institute of Science, Bangalore, 1995.

[Sharkey 1997] Noel E. Sharkey, *The New Wave in Robot Learning*, Department of Computer Science, University of Sheffield, U.K, 1997.

[Síma 1998] Jirí Síma, *Introduction to Neural Networks*, Technical report nº V-755, Institute of Computer Science, Academy of Scinces of the Czech Republic, 1998.

[Sutton 1988] Richard S. Sutton, *Learning to Predict by the Methods of Temporal Differences*, 1988.

[Sutton 1992] Richard S. Sutton, *Reinforcement Learning Architectures*, GTE Laboratories Incorporated, 1992.

[Sutton and Barto 1988] Richard S. Sutton and Andrew G. Barto, *Course Notes Reinforcement Learning I II and III*, 1988.

[Todd 1992] Peter M. Todd, *The Animat Path to Intelligent Adaptive Behavior*, 1992.

[Touzet 1997] Claude Touzet, *Neural Reinforcement Learning for Behaviour Synthesis*, 1997.

[Touzet 1998] Claude Touzet, *Bias Incorporation in Robot Learning*, 1998.

# Apéndice A: Definiciones

***Accionador:*** es la parte del robot que permite modificar el estado del mismo o del entorno. Son las "extremidades y músculos" del mismo.

***Actuador:*** ver accionador.

***CCD:*** (*Charge Coupled Device*) dispositivo de carga emparejada, instrumento cuyos semiconductores están conectados de tal manera que la salida (output) de uno sirve como entrada (*input*) del próximo. Es utilizado en cámaras digitales.

***CMOS:*** (*Complementary Metal Oxide Semiconductor*) tipo de semiconductor ampliamente utilizado, que contiene los parámetros de configuración del ordenador.

***Efectores:*** ver accionador.

***Función de recompensa (en aprendizaje por refuerzo):*** es la función que define la meta del aprendizaje; define qué eventos son buenos y cuales malos.

***Función de valor (en aprendizaje por refuerzo):*** es la que especifica qué es bueno en el largo plazo. Para esto intenta predecir la recompensa, pero es distinta de la función recompensa: por ejemplo, cuando jugamos ajedrez, poner a nuestro oponente en jaque mate está asociado con una gran recompensa, pero "comerle" su reina está asociado con un gran valor. La primera define la meta real de la tarea –ganar el juego- mientras que la segunda solamente predice la obtención de la meta.

***Inteligencia***: es la parte computacional de la habilidad para lograr metas en el mundo. Debido a que nuestra comprensión sobre los mecanismos de la inteligencia es limitada, todavía hay problemas para delimitar qué procedimientos se consideran inteligentes y cuáles no.

***Performance (aplicado a aprendizaje en robots):*** es cuán bueno es el desempeño obtenido. Dado un conjunto de metas se puede definir una medida cuantitativa de la performance del robot tal como la proporción de veces que el robot alcanza el objetivo ó como la suma de las recompensas que recibe a lo largo del tiempo.

***Política (en Aprendizaje por refuerzo):*** es la función que especifica la acción que debe tomarse ante cada situación[26] que se puede encontrar.

***Redes Neuronales Artificiales[27] (RNA):*** basan su comportamiento en una serie de elementos de procesamiento individuales, los cuales ejecutando una serie de cálculos aritméticos en forma conjunta, generan patrones numéricos que se asocian a la información que éstos reciben del medio ambiente -imágenes, sonido, texto- con el objeto de lograr una caracterización específica de tal información. Dichos patrones

[26] Aunque para la mayoría de los algoritmos no es necesario que las situaciones correspondan a estados, en toda la literatura se asume que las situaciones son mapeadas a ellos y se modelan las interacciones agente-entorno como una MDP.

[27] En [Síma 1998] hay una muy buena introducción al tema Redes Neuronales

serán almacenados y utilizados cada vez que las Redes Neuronales enfrenten nuevamente información permitiendo así el reconocimiento de la información actualmente cotejada, la generación de nuevas caracterizaciones o asociaciones nuevas. Bajo ésta mecánica las RNAs logran de manera aproximada:

- Aprendizaje Adaptativo, que es la habilidad de aprender a realizar tareas en base a ejemplos por entrenamiento;
- Auto-organización, una RNA puede crear su propia organización o representación de la información que recibe durante el tiempo de aprendizaje;
- Operación en tiempo real, una RNA es capaz de llevar a cabo cálculos en paralelo;
- Tolerancia a fallas, la destrucción parcial de una porción de la Red no impide el continuo funcionamiento de la misma.

***Robot:*** son dispositivos compuestos de sensores que reciben datos de entrada y que pueden estar conectados a la computadora. Ésta, al recibir la información de entrada, ordena al robot que efectúe una determinada acción. Puede ser que los propios robots dispongan de microprocesadores que reciben la lectura de los sensores y que estos microprocesadores ordenen al robot la ejecución de las acciones para las cuales está concebido.

***Robótica***: es la disciplina que se encarga del estudio y desarrollo de los robots. La robótica se centró, en primer lugar, en los robots de la llamada **primera generación**; es decir, incapaces de detectar los estímulos procedentes del entorno, limitados a funciones con una secuencia predeterminada y fija, siendo aplicados principalmente a la industria. Estos robots han dado paso a los que constituyen la **segunda generación**, capaces de desarrollar algún tipo de actividad sensorial. Los prototipos multisensoriales que interactúan en un grado muy elevado con el entorno se agrupan en la **tercera generación**. Para ello, la robótica se sirve de disciplinas como la mecánica, la microelectrónica y la informática, además de incorporar a los ingenios técnicas como el reconocimiento y análisis digital de las imágenes, el desarrollo de sistemas sensoriales, etc.

***Sensor:*** dispositivo utilizado por el robot para obtener información acerca del estado del mundo. Mide una característica del entorno y crea una señal eléctrica proporcional.

***Subsumpción*:** incluir en una categoría.

# Apéndice B: Comparación de las dos soluciones para el robot recolector.

Se realizaron 10 ejecuciones de cada solución en 9 posiciones diferentes.

| **Comparación (150,150)** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **promedio** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Agrupamiento fino** | ***Iteraciones*** | 3656 | 1605 | 2083 | 2703 | 6198 | 9324 | 4857 | 7043 | 2686 | 4279 | 4443 |
| | ***Premios*** | 125 | 119 | 130 | 132 | 128 | 74 | 105 | 133 | 92 | 98 | 114 |
| | ***Castigos*** | 3 | 0 | 0 | 3 | 4 | 18 | 1 | 0 | 0 | 38 | 7 |
| | ***Recompensa Promedio*** | 31.9 | 34.5 | 31.58 | 30.8 | 31.24 | 14.88 | 38.99 | 30.8 | 44.72 | 8.61 | 29.8 |
| **Dos Instintos** | ***Iteraciones*** | 746 | 2098 | 227 | 417 | 212 | 3057 | 179 | 557 | 506 | 556 | 856 |
| | ***Premios*** | 93 | 100 | 90 | 101 | 88 | 105 | 81 | 91 | 98 | 85 | 93 |
| | ***Castigos*** | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 2 | 0 | 1 | 1 |
| | ***Recompensa Promedio*** | 42.9 | 38.93 | 45.66 | 40.21 | 46.73 | 33.12 | 47.23 | 40.72 | 40.14 | 43.8 | 42 |
| **Comparación (600,300)** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **promedio** |
| **Agrupamiento fino** | ***Iteraciones*** | 1044 | 3497 | 3764 | 1650 | 2818 | 4383 | 12499 | 4963 | 359 | 4309 | 3303 |
| | ***Premios*** | 36 | 35 | 43 | 45 | 39 | 38 | 64 | 36 | 33 | 38 | 38 |
| | ***Castigos*** | 0 | 0 | 0 | 7 | 0 | 0 | 16 | 0 | 0 | 0 | 2 |
| | ***Recompensa Promedio*** | 37 | 38 | 30 | 25 | 33 | 35 | 15 | 37 | 41 | 35 | 33 |
| **Dos Instintos** | ***Iteraciones*** | 6572 | 4194 | 962 | 8000 | 4075 | 1427 | 4106 | 4698 | 1743 | 113 | 3364 |
| | ***Premios*** | 72 | 52 | 29 | 69 | 58 | 40 | 59 | 53 | 39 | 29 | 50 |
| | ***Castigos*** | 10 | 2 | 1 | 7 | 4 | 3 | 6 | 11 | 2 | 0 | 5 |
| | ***Recompensa Promedio*** | 16 | 22 | 37 | 18 | 22 | 30 | 20 | 21 | 30 | 49 | 27 |
| **Comparación (600,900)** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **promedio** |
| **Agrupamiento fino** | ***Iteraciones*** | 648 | 1069 | 831 | 185 | 479 | 1336 | 791 | 1217 | 821 | 3164 | 1054 |
| | ***Premios*** | 88 | 89 | 85 | 78 | 88 | 88 | 90 | 86 | 85 | 121 | 90 |
| | ***Castigos*** | 4 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 21 | 3 |
| | ***Recompensa Promedio*** | 36.04 | 36.9 | 38.8 | 42.63 | 36.79 | 37.33 | 36.67 | 38.47 | 39.13 | 23.43 | 36.62 |
| **Dos Instintos** | ***Iteraciones*** | 3192 | 791 | 864 | 1910 | 950 | 727 | 2390 | 870 | 2788 | 940 | 1542 |
| | ***Premios*** | 78 | 79 | 75 | 84 | 76 | 69 | 75 | 76 | 86 | 75 | 77 |
| | ***Castigos*** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | ***Recompensa Promedio*** | 42.29 | 40.5 | 41.24 | 39.69 | 42.38 | 44.64 | 44 | 43.51 | 38.97 | 42.21 | 41.94 |

| Comparación (250,900) | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agrupamiento fino | ***Iteraciones*** | 1979 | 14972 | 8282 | 15339 | 4600 | 2960 | 4572 | 3582 | 3079 | 3032 | 6240 |
| | ***Premios*** | 112 | 178 | 161 | 173 | 135 | 136 | 137 | 123 | 119 | 122 | 140 |
| | ***Castigos*** | 7 | 68 | 58 | 44 | 10 | 4 | 26 | 22 | 6 | 8 | 25 |
| | ***Recompensa Promedio*** | 32.66 | 15.88 | 17.78 | 17.85 | 27.22 | 27.7 | 23.74 | 26.77 | 31.23 | 29.72 | 25 |
| Dos Instintos | ***Iteraciones*** | 595 | 1252 | 325 | 709 | 310 | 596 | 583 | 261 | 216 | 415 | 526 |
| | ***Premios*** | 85 | 87 | 93 | 105 | 83 | 99 | 100 | 77 | 83 | 89 | 90 |
| | ***Castigos*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | ***Recompensa Promedio*** | 45.15 | 45.01 | 41.58 | 37.21 | 46.72 | 38.76 | 39.1 | 45.49 | 47.08 | 41.58 | 42.77 |

**Observación:** En la comparación (250,900), para el caso de la solución con un agrupamiento más fino, hubo 5 casos donde el robot se salió del escenario. Esto ocurrió porque esta posición no fue explorada suficientemente todavía (eso explica también que la solución con dos instintos sea tan exageradamente mejor -doce veces más rápida-).

| Comparación (250, 250) | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agrupamiento fino | ***Iteraciones*** | 3792 | 9146 | 4958 | 13782 | 527 | 12172 | 18306 | 6506 | 1699 | 3913 | 8253 |
| | ***Premios*** | 87 | 91 | 86 | 101 | 86 | 123 | 233 | 90 | 111 | 86 | 96 |
| | ***Castigos*** | 5 | 19 | 6 | 28 | 2 | 37 | 181 | 10 | 31 | 0 | 15 |
| | ***Recompensa Promedio*** | 29.49 | 24.91 | 29.64 | 21.45 | 30.58 | 17.07 | 6.72 | 27.32 | 19.11 | 31.45 | 25.67 |
| Dos Instintos | ***Iteraciones*** | 690 | 1236 | 92 | 4570 | 2116 | 3546 | 115 | 2283 | 3664 | 3701 | 2202 |
| | ***Premios*** | 63 | 74 | 57 | 87 | 74 | 81 | 59 | 76 | 83 | 82 | 74 |
| | ***Castigos*** | 0 | 4 | 0 | 7 | 11 | 3 | 0 | 5 | 5 | 6 | 4 |
| | ***Recompensa Promedio*** | 41.51 | 32.25 | 46.67 | 26.01 | 25.59 | 29.93 | 46.56 | 29.09 | 27.72 | 29.67 | 33.5 |

**Observación:** para el caso de dos instintos hubo ocasiones que resolvió el problema en 92 iteraciones o en 115. Esto se debió a que la posición inicial del robot fue en esos dos casos excelente (no tuvo que rodear sino simplemente empujar).

| Comparación (750, 410) | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agrupamiento fino | *Iteraciones* | 2093 | 6404 | 3635 | 5526 | 4121 | 4456 | 3204 | 578 | 4338 | 1417 | 3599 |
| | *Premios* | 54 | 45 | 54 | 54 | 55 | 74 | 50 | 59 | 55 | 57 | 56 |
| | *Castigos* | 4 | 0 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 | 2 |
| | *Recompensa Promedio* | 31.78 | 41.24 | 34.63 | 34.26 | 33.55 | 19.67 | 36.98 | 31.15 | 33.75 | 32.14 | 34.39 |
| Dos Instintos | *Iteraciones* | 621 | 100 | 90 | 248 | 107 | 787 | 1649 | 162 | 295 | 109 | 416 |
| | *Premios* | 45 | 40 | 40 | 40 | 39 | 39 | 38 | 42 | 40 | 40 | 40 |
| | *Castigos* | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| | *Recompensa Promedio* | 40.98 | 46.15 | 46.9 | 46.03 | 48.26 | 38.77 | 38.16 | 44.05 | 46.75 | 44.35 | 44.04 |
| **Comparación (910, 600)** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **promedio** |
| Agrupamiento fino | *Iteraciones* | 1103 | 15155 | 300 | 252 | 251 | 783 | 446 | 196 | 7406 | 361 | 2625 |
| | *Premios* | 72 | 90 | 73 | 70 | 71 | 90 | 85 | 73 | 99 | 77 | 76 |
| | *Castigos* | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 1 |
| | *Recompensa Promedio* | 46.96 | 37.19 | 46.43 | 48.94 | 47.9 | 38.35 | 38.8 | 46.52 | 33.6 | 44.06 | 44.75 |
| Dos Instintos | *Iteraciones* | 2135 | 2274 | 1905 | 2787 | 630 | 1071 | 1991 | 1635 | 1529 | 2183 | 1814 |
| | *Premios* | 92 | 80 | 74 | 80 | 79 | 76 | 85 | 93 | 74 | 74 | 81 |
| | *Castigos* | 3 | 1 | 2 | 3 | 0 | 0 | 1 | 7 | 2 | 0 | 2 |
| | *Recompensa Promedio* | 33.86 | 40.5 | 40.2 | 38.94 | 42.57 | 44.36 | 39.8 | 32.76 | 41.21 | 41.71 | 39.59 |

**Observación:** en la comparación (910,600), para el caso del agrupamiento fino, en un principio no tomamos en cuenta los experimentos que demoraron 7406 ni 15155 ya que se alejan mucho del resto de los valores afectando fuertemente el promedio. Como no los habíamos considerado, el valor era 462 iteraciones -en lugar de 2625-, por lo cual concluíamos que la solución con agrupamiento fino era cuatro veces más veloz que la que usa dos instintos. Esto nos hizo pensar que seguramente no se trate de desviaciones y que no sería correcto no considerar estos dos valores por ser muy grandes. Resultaría muy extraño que en el resto de los experimentos el comportamiento con dos instintos sea igual o mejor al obtenido con el agrupamiento más fino pero que en este caso sea cuatro veces peor.

| Comparación (120, 750) | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agrupamiento fino | ***Iteraciones*** | 6607 | 22176 | 3377 | 7672 | 2672 | 6746 | 1291 | 4050 | 905 | 1004 | 3813 |
| | ***Premios*** | 174 | 280 | 113 | 170 | 114 | 146 | 116 | 168 | 114 | 112 | 136 |
| | ***Castigos*** | 42 | 146 | 56 | 26 | 6 | 12 | 1 | 28 | 4 | 0 | 19 |
| | ***Recompensa Promedio*** | 17.09 | 8.65 | 4.58 | 18.8 | 30.86 | 23.95 | 31.69 | 18.92 | 31.74 | 33.28 | 23.43 |
| Dos Instintos | ***Iteraciones*** | 1338 | 1559 | 236 | 344 | 888 | 726 | 696 | 713 | 658 | 560 | 603 |
| | ***Premios*** | 85 | 107 | 77 | 83 | 98 | 87 | 99 | 92 | 88 | 93 | 135 |
| | ***Castigos*** | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | ***Recompensa Promedio*** | 42.44 | 31.09 | 47.31 | 44.46 | 37.66 | 42.86 | 37.49 | 41.1 | 40.99 | 40.02 | 40.54 |

**Observación:** para el promedio en el agrupamiento fino no consideramos el experimento con 22176 iteraciones ya que se desvía demasiado del resto de los valores. Si lo consideráramos el promedio pasa de 3813 a 5650. Para el promedio en la solución con dos instintos no consideramos los casos con 1338 ni 1559 iteraciones por el mismo motivo. Si los consideráramos el promedio pasa de 603 a 772.

| Comparación (400,400) | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | promedio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agrupamiento fino | ***Iteraciones*** | 1718 | 17036 | 1042 | 563 | 9811 | 61107 | 1608 | 298 | 2235 | 1376 | 3965 |
| | ***Premios*** | 20 | 29 | 15 | 14 | 28 | 28 | 26 | 19 | 26 | 27 | 23 |
| | ***Castigos*** | 0 | 9 | 0 | 0 | 10 | 14 | 6 | 0 | 9 | 7 | 5 |
| | ***Recompensa Promedio*** | 30.35 | 16.27 | 42.87 | 46.43 | 16.66 | 15.19 | 19.59 | 32.11 | 17.86 | 18.09 | 25.54 |
| Dos Instintos | ***Iteraciones*** | 139 | 98 | 106 | 149 | 89 | 33 | 46 | 247 | 107 | 781 | 180 |
| | ***Premios*** | 14 | 17 | 16 | 16 | 16 | 13 | 13 | 15 | 18 | 25 | 14 |
| | ***Castigos*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| | ***Recompensa Promedio*** | 46.57 | 36.53 | 37.94 | 31.2 | 38 | 48.23 | 47 | 42 | 33.72 | 24.23 | 40 |

**Observación:** En el caso del agrupamiento fino no consideramos el caso que demoró 61107 iteraciones, sino el promedio pasaría de 3965 a 9679. Esto resulta demasiado lento dada la cercanía de esta posición al nido (500, 500). Tal vez sea porque el cuadrante 1 no está lo suficientemente explorado. Además, esta posición está muy cerca de las fronteras entre los cuadrantes lo que ocasiona situaciones donde el robot está en un cuadrante y el objeto en otro. Éstas no fueron consideradas especialmente en el modelado y entorpecen lo aprendido. También observamos que el comportamiento con lo aprendido hasta el momento es empujar hacia el cuadrante superior derecho y allí se encuentra con otra situación no explorada lo suficiente. Esto explica el aparente mal comportamiento frente a una posición tan cercana al nido.

# Apéndice C1

## *Código fuente de la aplicación que sigue las paredes:*

(sólo las funciones más importantes, todo el archivo sim.cpp tiene un largo de 732 líneas entre el código del algoritmo *q-learning*, código del simulador, comentarios y espaciado)

```
/*variables*/
double beta;
double EXPLOTATION_THRESHOLD; /* explotation level, si es mayor explota */
int olds;
int nroIteraciones, nroQ[SIZE][NUM_ACTIONS];
float umbral=0.1,qvalues[SIZE][NUM_ACTIONS];
int cantrew=0, cantcast=0;

void init_qvalues()
{ FILE *f,*fn;
  int i,a,nro;
  float j;
  char resp;

  nroIteraciones=-1;

/*abrir el archivo "qval.txt" con los qvalues, esto permite guardar los
qvalues entre corrida y corrida si el aprendizaje se llega a interrumpir
porque el robot se salió de la arena. Y también abrir el archivo que contiene
la cantidad de veces que se visitó cada par (estado, acción) ya que el
algoritmo usa este valor para ir disminuyendo el valor del parámetro β */


  if ( (fn=fopen("nroQ.txt","rt"))==NULL || (f=fopen("qval.txt","rt"))==NULL){
       printf("No se pudo abrir uno de los archivos, Comienzo de Aprendizaje?:S o N ");
        scanf("%c",&resp);

        while (resp!='S' && resp!='s' && resp!='n' && resp!='N')
        {printf("Comienzo de Aprendizaje?: S o N ");
         scanf("%c",resp);
        }

        if (resp=='n' || resp=='N')
         {printf("Entonces provea los archivos necesarios.\n");
          exit(0);
        }
        else {nroIteraciones=0; //inicializo
              prom_recompensa=0;
              cant_recompensa_no_nula=0;
        }
   }
   else {fclose(f);
         fclose(fn);
   }

  if (nroIteraciones==0) //si es el comienzo de un aprendizaje inicializo
      {for (i=0;i<= (SIZE-1);i++)
            for (a=0;a<=(NUM_ACTIONS-1);a++)
            {qvalues[i][a]=0;
             nroQ[i][a]=1;
            }
  }
  else{
        if ( (f=fopen("qval.txt","rt"))==NULL ){
          printf("No se pudo abrir el archivo qval.txt ");
          exit(0);
        }
```

```
         if ( (fn=fopen("nroQ.txt","rt"))==NULL ){
            printf("No se pudo abrir el archivo nroQ.txt ");
            exit(0);
         }

        for (i=0;i<= (SIZE-1);i++)
           for (a=0;a<=(NUM_ACTIONS-1);a++)
              {fscanf(f,"%f",&j);
               qvalues[i][a]=j;
               fscanf(fn,"%i",&nro);
               nroQ[i][a]=nro;
         }
         fscanf(fn,"%i",&nroIteraciones);//cargo el nro de iteraciones
         fscanf(fn,"%f",&prom_recompensa);//cargo la recompensa promedio
         fscanf(fn,"%i",&cant_recompensa_no_nula);//cantidad para promediar

         fclose(f);
         fclose(fn);
       }

}
-----------------------------------------------

double best_qvalue(int state)  //devuelve el mayor qvalue de la mejor acción
para un estado
{
       double best_val = qvalues[state][0];
       int a;

       for (a=1;a<=(NUM_ACTIONS-1);a++)
         {
           if (qvalues[state][a] > best_val)
             {
              best_val = qvalues[state][a];
             }
         }
       return (best_val);
}
-----------------------------------------------
void update_qvalues(int news,int action,double r) //news es el estado actual
{/*esta función actualiza el valor del qvalue según el algoritmo, en base al
estado alcanzado y la recompensa obtenida*/

       double best_new_qval, qval;

       best_new_qval = best_qvalue(news);
         qval = qvalues[olds][action];
         nroQ[olds][action]= nroQ[olds][action] + 1;
         if (nroQ[olds][action]<8991)
             beta= 0.9-((double)nroQ[olds][action]/10000);  //reduzco el beta
         else beta=0.001;
       qvalues[olds][action] =(1 - beta)*qval + beta*(r + GAMMA*best_new_qval);
}
-----------------------------------------------
int choose_best_action(int state)
{/*elegir la mejor acción para un estado, explorando ó explotando según el
umbral de explotación*/
  double rvalue;
  float count,curr_val,best_val = -1000000.0,porcentaje_explotacion=0.05;
/*porcentaje_explotacion es el porcentaje mínimo al que baja el umbral
exploración explotación*/
  double aleatorio;
  int act,visitas=0,tope_visitas,tope_exp=1000;
/*tope_exp marca que a lo largo de 1000 iteraciones vamos disminuyendo el
umbral explotacion/exploración desde 1 hasta 0*/

  rvalue = (double)rand()/RAND_MAX;//0<=rvalue<1
```

```
  tope_visitas=(int)((1-porcentaje_explotacion)*tope_exp)+1;

  for (int i=0; i<NUM_ACTIONS; i++)
       visitas= visitas + nroQ[state][i];

int a;

printf("Visitas: %i\n",visitas);//muestro en la pantalla

if (visitas<tope_visitas)
       EXPLOTATION_THRESHOLD= 1 - ((double)visitas/tope_exp);
 else EXPLOTATION_THRESHOLD=porcentaje_explotacion;

 if (rvalue < EXPLOTATION_THRESHOLD) //si es menor explora, sino explota
  {
    printf("\n Explora\n");
    aleatorio= (double)rand()/(RAND_MAX+1);
//RAND_MAX +1 para que nunca de aleatorio=1

    act= (int) ((double) NUM_ACTIONS * aleatorio);
// va de cero a NUM_ACTIONS-1

  }
  else{
       printf("\n Explota\n");
         //elijo la mejor
       for (int a=0;a<=(NUM_ACTIONS-1);a++)
       {
              curr_val = qvalues[state][a];
              if (curr_val > best_val)
              {
                     best_val = curr_val;
                     act = a;
              }
       }
     }
  return (act);
}
-----------------------------------------------

int establecer_estado(int robot){

float s0,s1,s2,s3,s4,s5,s6,s7;
int estado;

world->setInput(robot);//tomo el nuevo valor de los sensores

  s0= input[robot][0];
  s1= input[robot][1];
  s2= input[robot][2];
  s3= input[robot][3];
  s4= input[robot][4];
  s5= input[robot][5];
  s6= input[robot][6];
  s7= input[robot][7];

if ( s0>umbral && s0>=s1 && s0>=s2 && s0>=s3 && s0>=s4 && s0>=s5 && s0>=s6 &&
s0>=s7 )
             {estado=0;}
else if (s1>umbral && s1>s0 && s1>=s2 && s1>=s3 && s1>=s4 && s1>=s5 && s1>=s6
&& s1>=s7)
             {estado=1;}
else if ( s2>umbral && s2>s0 && s2>s1 && s2>=s3 && s2>=s4 && s2>=s5 && s2>=s6
&& s2>=s7)
             { estado=2;}
else if ( s3>umbral && s3>s0 && s3>s1 && s3>s2 && s3>=s4 && s3>=s5 && s3>=s6
&& s3>=s7 )
             {estado=3;}
```

```
else if ( s4>umbral && s4>s0 && s4>s1 && s4>s2 && s4>s3 && s4>=s5 && s4>=s6 &&
s4>=s7 )
            {estado=4;}
else if( s5>umbral && s5>s0 && s5>s1 && s5>s2 && s5>s3 && s5>s4 && s5>=s6 &&
s5>=s7 )
            {estado=5;}
else if ( s6>umbral && s6>s0 && s6>s1 && s6>s2 && s6>s3 && s6>s4 && s6>s5 &&
s6>=s7 )
            {estado=6;}
else if ( s7>umbral && s7>s0 && s7>s1 && s7>s2 && s7>s3 && s7>s4 && s7>s5 &&
s7>s6 )
            {estado=7;}
else estado=8;

return estado;
}

-----------------------------------------------

void getMotorQlearner(){
/*función que implementa el qlearning para el robot*/

  double rew, dist;
  float p1, p2,t1,t2,t1d,t2d;
  int news;
  int a, tipoChoque,objChocado;
  float l,r;

/*elegir la mejor acción según el estado en el que está el robot*/
  a = choose_best_action(olds);

  switch(a) {
  case 0: l=0.80;r=0.80; break; //avanzar
  case 1: l=0.4; r=0.65; break; //girar a la izquierda
  case 2: l=0.65; r=0.4; break; //girar a la derecha
  }

  double tdist;

  t1=world->robot[0]->getX();
  t2=world->robot[0]->getY();
  world->runStep(l,r,0); // lo hago reaccionar

  world->setInput(0);

  tipoChoque = world->checkCrash(0,&objChocado);//código para que no atraviese
paredes

  if(tipoChoque == WALL_CRASH)
  {    t1d=world->robot[0]->getX();
       t2d=world->robot[0]->getY();
       tdist=g_distPoint(t1,t2,t1d,t2d);
      world->robot[0]->setX(world->robot[0]->getOX());
      world->robot[0]->setY(world->robot[0]->getOY());
  }

  news= establecer_estado(0);

dist = g_distPoint(world->robot[0]->getOX(),world->robot[0]->getOY(),
                   world->robot[0]->getX(),world->robot[0]->getY());

//cálculo de la recompensa
 switch(news) {
  case 0:   switch(olds) {
             case 0: rew=0;break;
             case 1: rew=0;break;
             case 2: rew=0;break;
             case 3: rew=0;break;
```

```
            case 4: rew=0;break;
            case 5: rew=-1;break;//perdió el estado óptimo
            case 6: rew=0;break;
            case 7: rew=0;break;
            case 8: rew= 0;break;
          }
          break;
case 1:   switch(olds) {
            case 0: rew=0;break;
            case 1: rew=0;break;
            case 2: rew=0;break;
            case 3: rew=0;break;
            case 4: rew=0;break;
            case 5: rew=-1;break;//perdió el estado óptimo
            case 6: rew=0;break;
            case 7: rew=0;break;
            case 8: rew=0;break;
            }
          break;
case 2:   switch(olds) {
            case 0: rew=0;break;
            case 1: rew=0;break;
            case 2: rew=0;break;
            case 3: rew=0;break;
            case 4: rew=0;break;
            case 5: rew=-1;break; //perdió el estado óptimo
            case 6: rew=0;break;
            case 7: rew=0;break;
            case 8: rew=0;break;
            }
          break;
case 3:   switch(olds) {
            case 0: rew=0;break;
            case 1: rew=0;break;
            case 2: rew=0;break;
            case 3: rew=0;break;
            case 4: rew=0;break;
            case 5: rew=-1;break;//perdió el estado óptimo
            case 6: rew=0;break;
            case 7: rew=0;break;
            case 8: rew=0;break;
        }
        break;

case 4:   switch(olds) {
            case 0: rew=0;break;
            case 1: rew=0;break;
            case 2: rew=0;break;
            case 3: rew=0;break;
            case 4: rew=0;break;
            case 5: rew=-1;break;//perdió el estado óptimo
            case 6: rew=0;break;
            case 7: rew=0;break;
            case 8: rew=0;break;
        }
        break;
case 5:   switch(olds) {
            case 0: rew=+1;break;
            case 1: rew=+1;break;
            case 2: rew=+1;break;
            case 3: rew=+1;break;
            case 4: rew=+1;break;
            case 5: rew=+10*dist;break;
            case 6: rew=+1;break;
            case 7: rew=+1;break;
            case 8: rew=+1;break;
        }
        break;
```

```
  case 6:   switch(olds) {
             case 0: rew=0;break;
             case 1: rew=0;break;
             case 2: rew=0;break;
             case 3: rew=0;break;
             case 4: rew=0;break;
             case 5: rew=-1;break;//perdió el estado óptimo
             case 6: rew=0;break;
             case 7: rew=0;break;
             case 8: rew=0;break;
          }
          break;

  case 7:   switch(olds) {
             case 0: rew=0;break;
             case 1: rew=0;break;
             case 2: rew=0;break;
             case 3: rew=0;break;
             case 4: rew=0;break;
             case 5: rew=-1;break;
             case 6: rew=0;break;
             case 7: rew=0;break;
             case 8: rew=0;break;
          }
          break;

  case 8:   switch(olds) {
             case 0: rew=-1;break;
             case 1: rew=-1;break;
             case 2: rew=-1;break;
             case 3: rew=-1;break;
             case 4: rew=-1;break;
             case 5: rew=-1;break;
             case 6: rew=-1;break;
             case 7: rew=-1;break;
             case 8: rew=0;break;
          }
          break;
  }
       printf("Estado:%i ",news);
       if (rew>0) {printf("premio;\n");cantrew++;}
       else if (rew<0) {printf("castigo;");cantcast++;}

  update_qvalues(news,a,rew);

      if (nroIteraciones==100000) //termino la ejecución en 100.000 iteraciones
            {
             FILE *f_out=fopen("qvalSal.txt","wt"),
*fn_out=fopen("nroQSal.txt","wt");
              printf("#reward: %i; #castigo: %i " ,cantrew, cantcast);
              for (int i=0;i<= (SIZE-1);i++)
                       for (a=0;a<=(NUM_ACTIONS-1);a++)
                {fprintf(f_out,"%f ",qvalues[i][a]);
                 fprintf(fn_out,"%i ",nroQ[i][a]);
                }

            fclose(f_out);
            fclose(fn_out);
            exit(0);
            }
       else   olds = news;
}

-----------------------------------------------
```

```
int main(int argc,char **argv){

  char worldFileStr[200];
   int run;

   seed=time(NULL);
   srand(seed);

. . . . . .

   world->setInput(0);
   init_qvalues();

/*determino el estado inicial*/
   olds= establecer_estado(0);
   seed=time(NULL);
   srand(seed);

   if (olds ==8){
              world->runStep(0.8,0.3,0); // lo hago girar a la derecha

              while (establecer_estado(0)==8)//sigue sin encontrar una pared
              {      printf("\n Instinto\n");
                     dibujar();
                     world->robot[0]->saveState();
                     world->runStep(0.8,0.8,0); // lo hago avanzar
                     tipoChoque = world->checkCrash(0,&objChocado);

                     if(tipoChoque == WALL_CRASH)
                     {      world->robot[0]->setX(world->robot[0]->getOX());
                            world->robot[0]->setY(world->robot[0]->getOY());
                     }
              }
              olds= establecer_estado(0);
       }

     for(;;){
              nroIteraciones++;
              printf("Iteracion: %i ; Old:%i ",nroIteraciones,olds);

              if (olds !=8)getMotorQlearner();
              else instinto(0);//si perdió la pared uso un instinto

              world->robot[0]->saveState();
              dibujar();
     }
}
```

## *Código fuente generado por Programación Genética:*

(lamentablemente no aparece el código del programa necesario para implementar la solución al problema de seguir paredes, sino uno de los programas generados a partir de la programación genética, extraido de [Dain 1998])

```
0 WhileTooFarFromWall
1    WhileTooFarFromWall
2 Do2
3    Do2
4 WhileTooCloseToWall
5    WhileInCoridorRange
6 TurnParallelToClosestWall
7 Do2
8            WhileInCoridorRange
9               Do2
10                WhileTooCloseToWall
11                   WhileTooFarFromWall
12                      MoveForward
13                Do2
14                   Do2
15                      WhileTooCloseToWall
16                         TurnParallelToClosestWall
17                      Do2
18                         Do2
19                            WhileTooCloseToWall
20                               WhileInCoridorRange
21                                  TurnParallelToClosestWall
22                            Do2
23                               WhileInCoridorRange
24                                  Do2
25                                     WhileTooCloseToWall
26                                        WhileTooFarFromWall
27                                           MoveForward
28                                     Do2
29                                        Do2
30                                           WhileTooCloseToWall
31                                              TurnParallelToClosestWall
32                                           Do2
33                                              MoveForward
34                                              IfConvexCorner
35                                                 TurnTowardsClosestWall
36                                        Do2
37                                           MoveForward
38                                           WhileInCoridorRange
39                                              Do2
40                                                 TurnTowardsClosestWall
41                                                 Do2
42                                                    TurnParallelToClosestWall
43                                                       MoveForward
44                               MoveForward
45                            WhileInCoridorRange
46                               TurnAwayFromClosestWall
47                      Do2
48                         MoveForward
49                         WhileInCoridorRange
50                         Do2
51                            TurnTowardsClosestWall
52                            Do2
53                               TurnParallelToClosestWall
54                               MoveForward
55            MoveForward
56         WhileInCoridorRange
57            TurnAwayFromClosestWall
```

# Apéndice C2

## *Código de la aplicación robot-recolector*

(sólo las funciones más importantes. Todo el archivo sim.cpp tiene un largo de 1014 líneas de código del algoritmo de *q-learning*, código para el simulador, comentarios y espaciado)

```
double beta;
double EXPLOTATION_THRESHOLD; /* explotation level, si es mayor explota */
int olds, cant_sin_premio,cant_recompensa_no_nula=0;
int nroIteraciones, nroQ[SIZE][NUM_ACTIONS];
float umbral=0.5,qvalues[SIZE][NUM_ACTIONS],prom_recompensa;
int cantrew=0, cantcast=0;
float Xnido=500,Ynido=500, Xmundo=1000, Ymundo=1000;
//las coordenadas Mundo son la parte útil del mismo, lejos de paredes

FILE *ft;
-----------------------------------------------
void dibujar(){
//código necesario para que se vea el movimiento del robot y el objeto

#ifdef GUI
if(gui_stepping){
#ifdef WIN32
       WaitForSingleObject(gui_step,INFINITE);
#else
       sem_wait(&gui_step);
#endif
}
#endif /* GUI */

#ifdef GUI
world->drawBots();
#endif /* GUI */
}
-----------------------------------------------
```

```
void init_qvalues()
{ FILE *f,*fn;
  int i,a,nro;
  float j;
  char resp;

nroIteraciones=-1; //para que luego no entre en el if

if ( (fn=fopen("nroQ.txt","rt"))==NULL || (f=fopen("qval.txt","rt"))==NULL){

       //permite comenzar una ejecución sin los archivos
       printf("No se pudo abrir uno de los archivos.Comienzo de Aprendizaje?:S o N ");

       scanf("%c",&resp);

       while (resp!='S' && resp!='s' && resp!='n' && resp!='N')
              {printf("Comienzo de Aprendizaje?: S o N ");
               scanf("%c",resp);
              }

       if (resp=='n' || resp=='N')
              {printf("Entonces provea los archivos necesarios.\n");
               exit(0);
       }
       else {nroIteraciones=0;
              prom_recompensa=0;
             cant_recompensa_no_nula=0;
        }
 }
 else {fclose(f);
       fclose(fn);
 }

 ft=fopen("graficar.txt","at");

if (nroIteraciones==0)
{for (i=0;i<= (SIZE-1);i++)
 for (a=0;a<=(NUM_ACTIONS-1);a++)
       {qvalues[i][a]=0;
        nroQ[i][a]=1;
       }
 }
else{
     if ( (f=fopen("qval.txt","rt"))==NULL ){
       printf("No se pudo abrir el archivo qval.txt ");
       exit(0);
     }

     if ( (fn=fopen("nroQ.txt","rt"))==NULL ){
       printf("No se pudo abrir el archivo nroQ.txt ");
       exit(0);
     }

     for (i=0;i<= (SIZE-1);i++)
       for (a=0;a<=(NUM_ACTIONS-1);a++)
       {fscanf(f,"%f",&j);
        qvalues[i][a]=j;
        fscanf(fn,"%i",&nro);
        nroQ[i][a]=nro;
       }
   fscanf(fn,"%i",&nroIteraciones);//cargo el nro de iteraciones entre ejec.
   fscanf(fn,"%f",&prom_recompensa);//cargo la recompensa promedio
   fscanf(fn,"%i",&cant_recompensa_no_nula);//cargo la cantidad para promediar
   fclose(f);
   fclose(fn);
}
}
-----------------------------------------------
```

```
double best_qvalue(int state)  {
...
//igual a la del apéndice C1
....
}
-----------------------------------------------

void update_qvalues(int news,int action,double r){
…
//igual a la del apéndice C1
…
}
-----------------------------------------------

int bien_posicionado(float sobstX, float sobstY, float robX, float robY)
{
if ( ((sobstX <= Xnido && robX <= sobstX) || (sobstX >= Xnido && robX >= sobstX))  &&
       ((sobstY <= Ynido && robY <= sobstY) || (sobstY >= Ynido && robY >= sobstY)) )
       return 1;
   else return 0;
}
-----------------------------------------------
int establecer_estado (int robot)
{      float s1,s2,s3,s4,s5,s6,at,t1,t2;
       double dir;
       int estado;

       world->setInput(robot);
       s1= input[robot][0];
       s2= input[robot][1];
       s3= input[robot][2];
       s4= input[robot][3];
       s5= input[robot][4];
       s6= input[robot][5];
       at= input[robot][6]+input[robot][7];

       if ( s1>=umbralPoco && s1>=s2 && s1>=s3 && s1>=s4 && s1>=s5 && s1>=s6 && s1>=at)
              { if (s1<umbralMucho) estado=1;
                else estado=2;
              }
       else   if (s2>=umbralPoco && s2>s1 && s2>=s3 && s2>=s4 && s2>=s5 && s2>=s6 && s2>=at)
              { if (s2<umbralMucho) estado=3;
                else estado=4;
              }
       else   if (s3>=umbralPoco && s3>s1 && s3>s2 && s3>=s4 && s3>=s5 && s3>=s6 && s3>=at)
              { if (s3<umbralMucho) estado=5;
                else estado=6;
              }
       else   if (s4>=umbralPoco && s4>s1 && s4>s2 && s4>s3 && s4>=s5 && s4>=s6 && s4>=at)
              { if (s4<umbralMucho) estado=7;
                else estado=8;
              }
       else   if (s5>=umbralPoco && s5>s1 && s5>s2 && s5>s3 && s5>s4 && s5>=s6 && s5>=at)
              { if (s5<umbralMucho) estado=9;
                else estado=10;
              }
       else   if (s6>=umbralPoco && s6>s1 && s6>s2 && s6>s3 && s6>s4 && s6>s5 && s6>=at)
              { if (s6<umbralMucho) estado=11;
                else estado=12;
              }
       else   if (at>=umbralPoco && at>s1 && at>s2 && at>s3 && at>s4 && at>s5 && at>s6)
              {estado=13;
              }
       else estado=0;

       dir= world->robot[robot]->direction;

       if (dir>=0 && dir<90)
```

```
              estado= estado + (14*0); //lo dejo para que la fórmula se entienda
       else if (dir>=90 && dir<180)
              estado= estado + (14*1);
            else if (dir>=180 && dir<270)
                   estado= estado + (14*2);
               else if (dir>=270 && dir<360)
                    estado = estado + (14*3);
                    else {printf("ERROR CON DIRECTION"); exit(0);}

       t1 = world->robot[robot]->getX() ;
       t2 = world->robot[robot]->getY();

       if (t1 <= Xnido && t2 <= Ynido)
        estado = estado + (56*0);
       else if (t1 > Xnido && t2 <= Ynido)
              estado = estado + (56*1);
            else if (t1 > Xnido && t2 > Ynido)
                    estado = estado + (56*2);
                 else estado = estado + (56*3);
       return estado;
}

-----------------------------------------------
void datos_para_graficar(float dist_al_home,float rew){

if (rew!=0) {
//solo tomo en cuenta las recompensas no nulas, sino me tiran el promedio a cero
       cant_recompensa_no_nula++;
prom_recompensa=((cant_recompensa_no_nula-1) * prom_recompensa+rew) /cant_recompensa_no_nula;
}

//imprime la recompensa promedio y la distancia al home del objeto
fprintf(ft, "%f\t%f\t",prom_recompensa,dist_al_home);

//imprime la cantidad de veces que se visitó cada estado
for (int i=0;i<SIZE; i++)
       { int visitas=0;

       for (int a=0;a<NUM_ACTIONS;a++)
              visitas= visitas + nroQ[i][a];

       fprintf(ft,"\t%i",visitas);
}
fprintf(ft,"\n");*/

}
-----------------------------------------------
```

```
void getMotorQlearner(){
  float p1, p2,t1,t2,t1d,t2d,l,r,delta_q;
  int a, tipoChoque,objChocado, salio=0,news;
  double arct,tdist, rew, dist_ant, dist_act;
  FILE *f_out, *fn_out;

  nroIteraciones++;
  a = choose_best_action(olds);
  if (a<0){
       printf("Accion menor a cero:%i",a);
       exit(0);
  }

  switch(a) {
       case 0: l=0.80;r=0.80;break;
       case 1: l=0.35;r=0.65;break;
       case 2: l=0.65;r=0.35;break;
       case 3: l=0.20;r=0.20;break;
  }

  p1 = world->sobst[0]->x;
  p2 = world->sobst[0]->y;
  dist_ant = g_distPoint(p1,p2,Xnido,Ynido);
  world->runStep(l,r,0); // lo hago reaccionar
  world->setInput(0);
  dibujar();
  tipoChoque = world->checkCrash(0,&objChocado);

  if (tipoChoque==SOBST_CRASH){   /*algo para que empuje al objeto*/
             world->robot[0]->carry=objChocado;
             world->moveSObst(0);
             dibujar();
  }

  news= establecer_estado(0);

  p1 = world->sobst[0]->x;
  p2 = world->sobst[0]->y;
  dist_act = g_distPoint(p1,p2,Xnido,Ynido);

  //función de recompensa
  if (dist_act!=dist_ant)
  {
        rew= int((dist_ant-dist_act)*10 );
        printf("Lo movio: %f unidades",dist_ant-dist_act);
  }
  else rew=0;

  if (rew>0) {printf("\n\n\t\tPREMIO\n");cantrew++;}
  else if (rew==0) printf("\nda igual\n");
       else {printf("\n\tCASTIGO\n\n");cantcast++;}

  world->setInput(0);

  /*si al final perdió al objeto lo marco con una recompensa neutra*/
 if(input[0][0]<umbralPoco && input[0][1]<umbralPoco && input[0][2]<umbralPoco
&& input[0][3]<umbralPoco && input[0][4]<umbralPoco && input[0][5]<umbralPoco
&& (input[0][6]+input[0][7])<umbralPoco) rew=0;

 //end función de recompensa

//actualizo
update_qvalues(news,a,rew);

datos_para_graficar(rew,dist_act);

t1 = world->robot[0]->getX() ;
t2 = world->robot[0]->getY();
```

```
/*controlo que no se haya ido del 'mundo útil':
(0,100)(1000,100)(1000,1000)(0,1000)*/
if (p1<0 || t1<0 || p1>1000 || t1>1000 || p2<100 || t2<100 || p2>1000 || t2>1000)
             salio=1;

if (dist_act<80 || salio==1)
{if (salio!=1)
             printf("Llego en Iteraciones=%i .",nroIteraciones);
 else printf("\nSe salio de la arena ... en %i Iteraciones",nroIteraciones);

 /*escribir q-values en un archivo y salir*/
 f_out=fopen("qvalSal.txt","wt");
 fn_out=fopen("nroQSal.txt","wt");

 for (int i=0;i<= (SIZE-1);i++)
       for (a=0;a<=(NUM_ACTIONS-1);a++)
       {fprintf(f_out,"%f ",qvalues[i][a]);
       fprintf(fn_out,"%i ",nroQ[i][a]);
       }

 fprintf(fn_out,"%i ",nroIteraciones);//guardo el nro de iteraciones
 fprintf(fn_out,"%f ",prom_recompensa); //guardo recompensa promedio
 fprintf(fn_out,"%i",cant_recompensa_no_nula); //cantidad de rec no nulas

 fclose(ft);
 fclose(f_out);
 fclose(fn_out);

 printf("#reward: %i; #castigo: %i " ,cantrew, cantcast);
 exit(0);
 }
 else olds = news;
}
-----------------------------------------------

void instinto(int robot){

int tipoChoque, objChocado;

       printf("\n Instinto\n");

       while (establecer_estado(robot)==6)//sigue sin encontrar una pared
       {      printf("\n Instinto\n");
              dibujar();
              world->robot[robot]->saveState();
              world->runStep(0.8,0.3,robot); // lo hago girar a la derecha
              world->runStep(0.8,0.8,robot); // lo hago avanzar
              tipoChoque = world->checkCrash(robot,&objChocado);

              if(tipoChoque == WALL_CRASH)
              {      world->robot[robot]->setX(world->robot[robot]->getOX());
                     world->robot[robot]->setY(world->robot[robot]->getOY());
              }
       }

       olds= establecer_estado(robot);

if (p1<0 || t1<0 || p1>1000 || t1>1000 || p2<100 || t2<100 || p2>1000 || t2>1000)
                     {//controlo que no se salga del mundo
                     printf("#reward: %i; #castigo: %i " ,cantrew, cantcast);
                     f_out=fopen("qvalSal.txt","wt");
                     fn_out=fopen("nroQSal.txt","wt");
                     for (int i=0;i<= (SIZE-1);i++)
                       for (a=0;a<=(NUM_ACTIONS-1);a++)
                     {      fprintf(f_out,"%f ",qvalues[i][a]);
                            fprintf(fn_out,"%i ",nroQ[i][a]);
                     }
```

```
                    fclose(f_out);
                    fclose(fn_out);
                    exit(0);
                    printf("#reward: %i; #castigo: %i " ,cantrew, cantcast);
        }
}
-----------------------------------------------

int main(int argc,char **argv){

  char worldFileStr[200];
  int run,i,a,tipoChoque, objChocado;
  FILE *f_out, *fn_out;
  float p1,p2,t1,t2;
  double dir,arct,arct_opuesto;
  int seed;

  /*comienza código para el simulador*/
  seed=time(NULL);
  srand(seed);

.....igual a la aplicación anterior....

/*fin del código específico para el simulador*/

 printf("starting trial...\n");
 init_qvalues();

/*determino el estado inicial*/
    olds= establecer_estado(0);

    for(;;){
//por siempre…. En realidad al llegar cerca del nido o salirse del 'mundo' se termina la simulación
      dibujar();
       printf("Iteracion: %i\n",nroIteraciones);
       /*guardo coordenadas del robot y del objeto (sobst)*/
       t1=world->robot[0]->getX();
       t2=world->robot[0]->getY();
       p1 = world->sobst[0]->x;
       p2 = world->sobst[0]->y;
       world->setInput(0);
//si pierde al objeto entonces instinto, sino q-learning
        if(input[0][0]<umbralPoco && input[0][1]<umbralPoco &&
input[0][2]<umbralPoco && input[0][3]<umbralPoco && input[0][4]<umbralPoco &&
input[0][5]<umbralPoco && (input[0][6]+input[0][7])<umbralPoco) {
       instinto(0);
       }
       else   getMotorQlearner();
    }
}
```