6 Comportamientos Símil-Insecto

Entonces debes hacerme saber
¿Debo quedarme o debo irme?
De la canción, *Debo quedarme o debo irme*, Mick Jones (The Clash), 1982.

Página opuesta: Vaquita de San Antonio
La foto es cortesía de Jon Sullivan (www.pdphoto.org)

Diseñar comportamientos para los robots es un desafío, pero es un proceso divertido. No hay una metodología formal o técnica a seguir. Implica creatividad, la habilidad para reconocer las fortalezas y las limitaciones del robot físico, el tipo de entorno en el cual el robot desarrollará su comportamiento, y por supuesto, el conocimiento de los paradigmas disponibles para programar comportamientos de robot. Ya han visto cómo, incluso un robot simple como el Scribbler, puede ser programado para llevar a cabo un diverso rango de comportamientos. También hemos ocupado un considerable esfuerzo hasta ahora explorando el abanico de posibles funciones que un robot puede realizar. El lugar en el que está el robot al funcionar, puede cumplir un rol importante al exhibir un comportamiento programado con éxito. En este capítulo, observaremos el comportamiento de los robots desde otro ángulo.

Vehículos Braitenberg

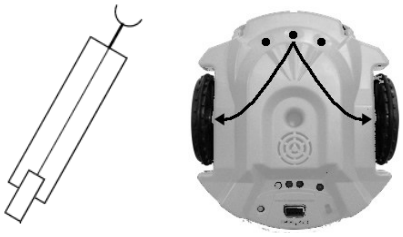
En 1984, Valentino Braitenberg escribió un libro titulado "*Vehículos: Experimentos en Psicología Sintética*" (MIT Press). En él describe varios comportamientos ideados que se centran en la creación de vehículos simples con mecanismos de control muy simples que muestran comportamientos aparentemente complejos. El objetivo de estos experimentos era ilustrar algunos aspectos esenciales de la estructura interna de los cerebros animales (y humanos). Cada experimento comprende la descripción de un vehículo simple que es provisto de un pequeño grupo de sensores (bastante similar al nuestro robot Scribbler) y el modo en que los sensores pueden estar conectados a los motores de estos vehículos imaginarios de forma análoga a las conexiones neurológicas en los animales. Muestra cómo los vehículos resultantes son capaces de realizar comportamientos complejos que pueden ser descriptos como: temor, agresión, amor, lógica, libre voluntad, etc.

Un tema central que subyace a los experimentos de Braitenberg es la demostración de lo que él llama la Ley del análisis cuesta arriba y la invención cuesta abajo: es mucho más difícil adivinar la estructura interna de una entidad con sólo observar su comportamiento, que efectivamente crear la estructura que desemboca en ese comportamiento. Es decir, intentar postular la estructura interna a partir de la pura observación es una tarea cuesta arriba (más difícil), mientras que intentar crear una entidad que exhibe cierto comportamiento es una tarea cuesta abajo (fácil). Mientras que todos los vehículos de Braitenberg eran imaginarios, y no fueron diseñados para ser efectivamente fabricados, a la gente le resultó un ejercicio divertido e intelectualmente interesante crearlos. Los robots personales como el Scribbler son excelentes plataformas para hacer esto y en lo que sigue del texto, describiremos algunos de los vehículos de Braitenberg (y de tipo-Braitenberg) y diseñaremos comportamientos para el robot basados en ellos.

Vehículo 1: Vivo

El primer vehículo que describe Braitenberg tiene un sensor y un motor. El valor transmitido por el sensor directamente alimenta al motor. Si el valor que se reporta al sensor es una cantidad variable (digamos la luz), el vehículo se moverá a una velocidad proporcional a la suma de cantidad detectada por el sensor.

Vehículo#1: Vivo



Arriba se presenta un esquema del vehículo. A fin de diseñar este vehículo usando al Scribbler, pueden usar el sensor de luz del centro y conectar lo que reporte directamente a ambos motores del robot. Es decir, la misma lectura de luz está controlando directamente ambos motores en la misma cantidad. Como ya han visto, hay muchas maneras distintas de especificar comandos de movimientos del motor para el Scribbler. Supongamos que el valor obtenido del sensor de luz central es C, pueden controlar ambos motores usando este valor, al utilizar este comando:

```
motors(C, C)
```

Alternativamente, también pueden usar el comando `forward:`

```
forward(C)
```

Ahora que conocemos la estructura interna de este vehículo, podemos escribir un programa que lo implemente. Pero antes de llegar hasta este punto, debemos resolver un pequeño tema de compatibilidad: los sensores de luz reportan valores en los rangos de 0..5000, mientras que los comandos de movimiento del motor toman valores en el rango de –1.0 a 1.0. En este ejemplo, solamente nos interesan los movimientos que oscilan dentro del rango de un detenimiento completo a una velocidad máxima hacia delante, por lo tanto los valores oscilan en un rango de 0.0 a 1.0. Debemos normalizar computacionalmente, o mapear los valores del sensor de luz en este rango. Un primer intento en este sentido sería escribir una función llamada `normalizar` que opera de la siguiente manera:

```
def normalizar(v):
  # normaliza v en el rango 0.0 a 1.0
```

Una vez que tenemos esta función, podemos escribir el comportamiento del vehículo de la siguiente manera:

```
def main():
   # vehiculo de Braitenberg #1: Vivo
   while True:
      L = getLight("center")
      forward(normalizar(L))

main()
```

Normalizar Valores del Sensor

Es hora de pensar en la tarea de la función `normalizar` (normalización). Dado un valor recibido del sensor de luz, debe transformarlo en un valor proporcional entre 0.0 y 1.0 de modo que cuanto más brillante sea la luz, más alto será el valor (ej., más cercano a 1.0). Vehículo#1 se mueve en proporción a la cantidad de luz que recibe. Es un buen momento para volver a la función `senses` de Myro para ver los valores reportados por los sensores de luz. Adelante, revisen esta función.

Luego de examinar los valores devueltos por los sensores de luz, pueden notar que reportan valores pequeños (menos de 50) para la luz brillante y valores más grandes (tanto como 3000) para la oscuridad. De alguna manera, podemos decir que el sensor de luz es en realidad un sensor de oscuridad; cuanto más oscuro está, más altos son los valores que reporta. Los sensores de luz son capaces de reportar valores entre 0 y 5000. Ahora, definitivamente podemos calibrar o normalizar usando estos valores usando la siguiente definición de `normalizar`:

```
def normalizar(v):
  # Normaliza v (en el rango 0..5000) a 0..1.0, inversamente
```

```
  return 1.0 - v/5000.0
```

Es decir, dividimos el valor del sensor de luz por su máximo valor y luego restamos esto de 1.0 (para la proporcionalidad inversa). Por lo tanto, un valor de luz más brillante, digamos un valor de 35, se normalizará como:

```
1.0 - 35.0/5000.0 = 0.9929
```

Si 0.9929 se envía a los motores (como en el programa de arriba), el robot se moverá a toda velocidad hacia delante. Computemos la velocidad del robot cuando hay total oscuridad. Cuando ponen un dedo en el centro del sensor, obtendrán valores dentro del rango de 2000-3000. Para 3000, la normalización sería:

```
1.0 - 3000.0/5000.0 = 0.40
```

El robot todavía se estará moviendo, aunque casi a media velocidad. Lo más probable es que estén operando el robot en una habitación con suficiente luz ambiental. Podrán notar que bajo condiciones de luz ambiental diurna, los valores reportados por los sensores de luz oscilan en el rango de 150-250. Al usar la normalización de arriba, obtendrán:

```
1.0 - 200.0/5000.0 = 0.9599
```

Esto es casi avanzar a toda velocidad hacia delante. Para experimentar el verdadero comportamiento del vehículo de arriba, debemos usar un esquema de normalización que tome en cuenta las condiciones de luz ambiental (variarán de habitación a habitación). Más adelante, asumamos que en condiciones de luz ambiental, veremos al robot responder a la fuente de luz que controlaremos. Una linterna funcionará bien. Entonces, para hacer que el robot sea apropiadamente sensible a la linterna bajo condiciones de luz ambiental, pueden escribir una versión mejorada de normalización como se muestra a continuación:

```
def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente
```

Es decir, la condición más oscura está representada por el valor de luz ambiental (`Ambiente`) y luego la normalización se hace con respecto a ese valor. Pueden establecer el valor ambiental a mano, o una mejora manera es que el robot perciba la luz ambiental cuando el programa es iniciado. Esta es la misma versión de normalización que vieron en el capítulo previo. Ahora saben cómo llegamos allí. El programa completo para Vehículo#1 se muestra abajo:

```
# vehiculo de Braitenberg #1: Vivo
from myro import *
initialize("com"+ask("Qué puerto?"))

Ambiente = getLight("center")

def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente

def main():
   #  vehiculo de Braitenberg #1: Vivo
```

```
   while True:
      L = getLight("center")
      forward(normalizar(L))
```

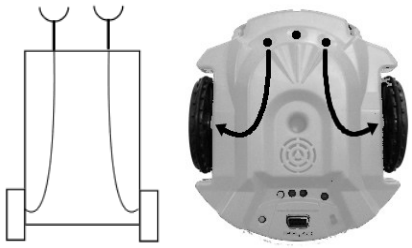
Realizar la siguiente actividad: Implementen el programa de arriba y observen el comportamiento del robot. ¿Responde como se describe arriba?

También habrán notado a esta altura que los tres sensores no necesariamente están sincronizados. Es decir, no reportan exactamente los mismos valores bajo las mismas condiciones. Al escribir programas para el robot que usan múltiples sensores de luz, es una buena idea promediar los valores devueltos por todos los sensores de luz para representar el valor ambiente. Modifiquen el programa de arriba para usar el promedio de los tres valores como el valor ambiente. No debería haber una diferencia notable en el comportamiento del robot. Sin embargo, esto es algo que querrán volver a usar en programas futuros.

Vehículo 2: Cobarde y Agresivo

El siguiente grupo de vehículos utiliza dos sensores. Cada sensor directamente maneja un motor. Por lo tanto, la velocidad del motor individual es directamente proporcional a la cantidad percibida por el sensor. Hay dos formas posibles de conectar el sensor. En el primer caso, Vehículo2a, el sensor a cada lado conecta con el motor del mismo lado. En el otro caso, Vehículo2b, las conexiones están intercambiadas. Es decir, el sensor izquierdo conecta con el motor derecho y el sensor derecho conecta con el motor izquierdo. Primero diseñemos el programa de control para el Vehículo 2a:

Vehículo 2a: Cobarde



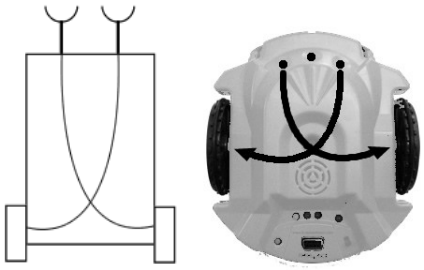
```
# vehiculo de Braitenberg #2: Cobarde
from myro import *
initialize("com"+ask("Qué puerto?"))

Ambiente = sum(getLight())/3.0

def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente

def main():
   #  vehiculo de Braitenberg #2: Cobarde
   while True:
      L = getLight("left")
      R = getLight("right")
      motors(normalizar(L), normalizar(R))
```

La estructura del programa de arriba es muy similar a la del Vehículo#1. Hemos modificado el establecimiento del valor de la luz ambiental al de un promedio de los valores de las tres luces. También usamos el comando `motors` para manejar el motor izquierdo y el derecho en forma proporcional a los valores de los sensores de luz izquierdo y derecho (luego de su normalización).

Realizar la siguiente actividad: Implementar el programa de control para el Vehículo 2a como se muestra arriba. Observen el comportamiento del robot alumbrando una linterna directamente frente al robot y en cada uno de los sensores izquierdo y derecho.

Luego, escriban el programa de control para el Vehículo2b como se muestra aquí. Esto requiere un cambio simple en el programa del Vehículo2a: cambiar los parámetros del comando `motors` para reflejar las conexiones intercambiadas. Nuevamente observen los comportamientos alumbrando la linterna directamente frente al robot y también un poco en cada lado.

Vehículo 2b: Agresivo



Notarán que los robots se comportan del mismo modo cuando la luz se ubica frente a ellos: Ambos están atraídos a la luz y por lo tanto se mueven hacia la fuente de luz. Sin embargo, el Vehículo2a se moverá alejándose de la luz si la fuente de luz está a un costado. Dado que el sensor más cercano se excitará más, moviendo al motor correspondiente más rápido, y por lo tanto alejando al robot. En el caso del Vehículo 2b, por otra parte, siempre girará hacia la fuente de luz y se moverá hacia ella. Braitenberg llama a estos comportamientos *cobarde* (2a) y *agresivo* (2b)

Controlando las Respuestas del Robot

Suele ser necesario, al diseñar y testear comportamientos de robot, generar apropiadamente el entorno y la orientación del robot dentro del mismo adecuadamente. En casos simples esto se logra fácilmente ubicando al robot en la orientación deseada y luego cargando y ejecutando el programa. Sin embargo, previamente al comportamiento efectivo del robot, quizás necesiten realizar algunas observaciones preliminares (por ejemplo, percibir la luz ambiental), es necesario reorientar apropiadamente al robot apropiadamente antes de iniciar la ejecución del comportamiento actual. Esto se puede lograr fácilmente incluyendo algunos comandos interactivos en el programa del robot. La estructura de programa resultante se muestra abajo:

```
# importar la librería myro, y establecer la conexión con el robot
# definir todas las funciones aquí (normalizar, etc.)
# establecer los valores para la condición de Ambiente

def main():
    # Descripción del comportamiento ...

    # Darle la oportunidad al usuario de acondicionar el robot
    askQuestion("Presione OK para comenzar...", ["OK"])

    # Escriba el comportamiento del robot aquí
```

Realizar la siguiente actividad: Modifiquen los programas de los vehículos 1, 2a y 2b para incluir el comando `askquestion` de arriba.

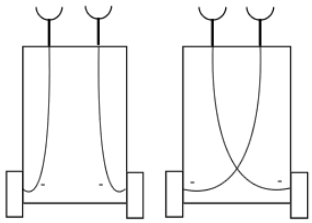
Hemos introducido algunos patrones básicos de programación arriba que pueden ser usados en muchas situaciones de programación. Lo que hay que recordar es que, en cualquier punto de la ejecución de un programa de robot, también pueden

programar apropiadamente interjecciones para realizar varias funciones experimentales o de control. Veremos varios ejemplos más adelante.

Otras Normalizaciones

Todas las normalizaciones de los valores de sensores de luz mostradas arriba fueron usadas para normalizar los valores dentro del rango de 0.0..1.0 en proporción directa a la cantidad de luz percibida. Es decir, cuanto más oscuro está, más cercanos estarán los valores normalizados a 0.0, y cuanto más brillo haya, más cerca estarán los valores normalizados a 1.0. Esta es sólo una forma en la que podemos relacionar la cantidad percibida con la cantidad de velocidad aplicada a los motores del robot. Pueden imaginarse otro tipo de relaciones. La más obvia, por supuesto, es la relación inversa: cuanto más oscuro está, más cerca de 1.0, y viceversa. Braitenberg llama a esto una relación *inhibitoria* (como opuesto al *excitatoria*): cuanto más se percibe una cantidad, más lento funcionan los motores del robot. Como en los vehículos 2a y 2b arriba, está la opción de dos tipos de conexiones: derecha y cruzada. Estos se muestran abajo (un signo más (+) junto a un conector indica una conexión excitatoria y un signo menos (-) representa una conexión inhibitoria):

Vehículos 3a (Amor) y 3b (Explorador)



Escribir la función `normalize` para una conexión inhibitoria es bastante directo:

```
def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return v/Ambiente
```

Braitenberg describe el comportamiento de los vehículos resultantes como *amor* (Vehículo 3a) y *explorador* (Vehículo 3b). Es decir, si observan el comportamiento de los dos vehículos, seguramente notarán que el Vehículo 3a se detendrá de frente a la fuente de luz (muy cerca), mientras que el Vehículo 3b se detendrá de espaldas a la fuente de luz y quizás se aleje, dependiendo de la presencia de otras fuentes de luz.

En otras variantes de normalizaciones de valores de sensor, Braitenberg sugiere usar funciones matemáticas no-monotónicas. Es decir, si observan las normalizaciones excitatorias e inhibitorias, pueden ser descriptas como monotónicas: más luz, mayor velocidad del motor; o más luz, menor velocidad del motor. Observen la función presentada en la página siguiente. La relación se incrementa en proporción a la entrada sensorial pero sólo hasta cierto punto y luego decrece. La incorporación de estas relaciones en los vehículos llevará a comportamientos más complejos (Braitenberg los describe como vehículos con *instintos*). A continuación se define una función de normalización, basada en la curvatura mostrada:

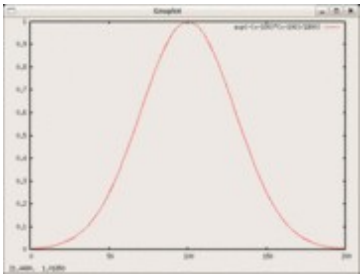
$$f(x) = e^{-(x-100)^2/1800}$$

La función de arriba está basada en una función más simple:

$$f(x) = e^{-x^2}$$

La primera definición está estirada para cubrir el rango 0..200 para valores de x con 100 como el punto donde reporta el máximo valor (por e. 1.0). Matemáticamente, generalmente esta función también se conoce como la *curva Gaussiana*. Una curva Gaussiana se define en términos de una *media* (π*) y desviación estándar* (σ) como se muestra abajo:

Una Función No-Monotónica



$$f(x) = e^{-(x-\pi)^2/2*\sigma^2}$$

Por lo tanto, en la función de normalización, estamos usando 100 como media y 30 como desviación estándar. Fácilmente pueden escalar la curva para el rango de los valores del sensor deseado usando la siguiente función `normalizar`:

```
def normalizar(v):
      mean = Ambiente/2.0
      stddev = Ambiente/6.0
        if v >= Ambiente:
                v = Ambiente
        return exp(-(v - mean)**2 / 2*(stddev**2))
```

`exp(x)` es una función Python que computa el valor de e^x . Está disponible en la librería `math` de Python. Ahondaremos en la librería math con más detalle en el siguiente capítulo. En función de usar la función `exp` como se muestra arriba, deben importar la librería `math`:

```
from math import *
```

Hay por supuesto varias otras posibilidades que pueden probarse: una función step, un umbral; y cualquier otra combinación matemática. La idea clave es que hay un mapeo claro del rango de los valores del sensor a los valores del motor en el rango de 0.0..1.0.

Los robots que usan estas normalizaciones y otras variaciones suelen mostrar comportamientos muy interesantes y a veces impredecibles. Observadores que desconozcan los mecanismos de mapeo internos tendrán dificultades en describir con precisión el comportamiento del robot y tenderán a usar términos antropomórficos (como *amor, odio, instintos*, etc.) para describir los comportamientos del robot. Esto es lo que significa un análisis cuesta arriba.

Sensores Múltiples

Agregar varios sensores enriquece el espacio de diseño para comportamientos del robot. Como diseñador, ahora tienen la posibilidad de elegir entre diferentes tipos de mapeos: excitatorio, inhibitorio, o más complejo; y entre conexiones: directa o cruzada. De pronto, el comportamiento de robot resultante puede parecer complejo. En el Scribbler, por ejemplo, además de los sensores de luz, también tienen el sensor de atascamiento y los sensores IR. Con la excepción de los sensores de luz, todos los otros sensores son digitales o de umbral que no son ni encendido ni apagado (ON, OFF) – por ejemplo, reportan valores que son 0 ó 1, indicando la presencia o ausencia de aquello que perciben. De alguna manera, se puede considerar que los sensores digitales ya están normalizados, pero aún es posible invertir esta relación y es necesario. Pueden diseñar varios comportamientos

interesantes combinando dos o más sensores y decidiendo si conectarlos en forma directa o cruzada.

Realizar la siguiente actividad: En su diseño de Vehículos 2a y 2b, sustituyan el sensor de obstáculos en lugar de los sensores de luz. Describan el comportamiento resultante de los vehículos. Intenten lo mismo para los Vehículos 3a y 3b. Luego, combinen el comportamiento de los vehículos resultantes con los sensores de luz. Prueben todas las combinaciones de conexiones, así como los mapeos inhibitorios y excitatorios. ¿Qué vehículos exhiben los comportamientos más interesantes?

Más Vehículos

Aquí hay descripciones de varios vehículos que tienen el espíritu de los diseños de Braitenberg y también exhiben comportamientos interesantes. Usando los conceptos y las técnicas de programación de arriba, intenten implementarlos en el robot Scribbler. Una vez completado, inviten a algunos amigos a observar los comportamientos de estas criaturas y registren sus reacciones.

Tímido

El tímido es capaz de moverse hacia delante en una línea recta. Tiene un sensor de luz de umbral que apunta hacia arriba. Cuando el sensor detecta luz, la criatura se mueve hacia delante, si no, se queda quieta. El umbral del sensor de luz debería estar establecido para luz ambiental. De esta manera, cuando la criatura “vea” la luz, se moverá. Al entrar en una sombra (que puede ser proyectada por una mano o cualquier otro objeto), se detendrá. Si lo que sea que está proyectando la sombra se mueve, la criatura volverá a moverse. Por eso, el tímido es un buscador de sombra.

Indeciso

El indeciso es similar al tímido, excepto que nunca se detiene: sus motores siempre están funcionando, ya sea hacia delante o hacia atrás, controlados por el sensor de luz de umbral. Cuando el detector de luz percibe luz, se mueve hacia adelante, si no, se mueve hacia atrás. Al hacer funcionar a esta criatura, notarán que tiende a oscilar hacia delante y hacia atrás en los bordes de la sombra. Por eso, el indeciso es un buscador de bordes de sombra.

Paranoico

El paranoico es capaz de girar. Esto se logra moviendo el motor derecho hacia delante y el motor izquierdo en dirección reversa al mismo tiempo. Tiene un solo sensor de luz de umbral. Cuando el sensor detecta luz, se mueve hacia delante. Cuando el sensor entra en la sombra, revierte la dirección del motor izquierdo, girando hacia la derecha. Pronto el sensor se dará vuelta y saldrá de la luz. Cuando esto suceda, volverá a su movimiento hacia delante. El paranoico es una criatura que teme a la sombra.

Esto, Eso, o lo Otro

La sentencia `if` introducida en el capítulo 5 es una forma de tomar decisiones simples. Es decir, pueden controlar condicionalmente la ejecución de un conjunto de comandos basado en una condición singular. La sentencia if en Python es bastante versátil y puede ser usada para tomar decisiones múltiples. Así es como deberían usarlo para elegir entre dos grupos de comandos:

```
if <condicion>:
```

```
  <esto>
else:
  <eso>
```

Es decir, si `<condicion>` es verdadero realizará los comandos especificados en `<exto>`. Sin embargo, si `<condicion>` es falso, hará `<eso>`. De manera similar, pueden extender la sentencia if para ayudar a especificar opciones múltiples:

```
if <condicion-1>:
  <esto>
elif <condicion-2>:
  <eso>
elif <condicion-3>:
  <otra cosa >
...
else:
  <otro>
```

Noten el uso de la palabra `elif` (¡sí, se deletrea así!) para designar “else if”. Entonces, dependiendo de la condición que sea verdadera, el correspondiente `<esto> o <eso>` u `<otra cosa>` se llevará a cabo. Si el resto falla, el `<otro>` será llevado a cabo.

Simples Comportamientos Reactivos

Al usar los tres sensores de luz, el robot puede detectar condiciones variables de luz en su entorno. Escribamos un programa de robot que lo hace detectar y orientarse hacia la luz brillante. Recuerden del capítulo 5 que los sensores de luz reportan valores bajos en condiciones de luz brillante y valores altos con luz baja. Para realizar esta tarea, solamente necesitamos mirar los valores reportados por los sensores de luz izquierdo y derecho. A continuación se describe el comportamiento del robot:

```
do para una cantidad de tiempo
  if la luz izquierda es más brillante que la derecha
    girar a la izquierda
  else
    girar a la derecha
```

De esta manera, haciendo uso de la sentencia-if, podemos refinar lo expresado arriba de la siguiente manera:

```
while timeRemaining(30):
  if la luz izquierda es más brillante que la derecha:
    turnLeft(1.0)
  else:
    turnRight(1.0)
```

Lo único que queda en los comandos de arriba es escribir la condición para detectar la diferencia entre los dos sensores de luz. Esto puede hacerse usando la expresión:

```
getLight('left') < getLight('right')
```

Realizar la siguiente actividad: Escriban un programa completo que implemente el comportamiento de arriba y pruébenlo en su robot.

Quizás hayan notado que aún en condiciones de luz uniformes tienden a reportar valores diferentes. Generalmente es una buena idea determinar un umbral de

diferencia al tomar la decisión de arriba. Digamos que definimos un umbral en una diferencia de por lo menos 50. Es decir que si los sensores izquierdo y derecho difieren en por lo menos 50, entonces, dobla hacia el sensor más brillante. ¿Qué ocurre si la diferencia es menor que la del umbral? Decidamos que en ese caso el robot se quedará quieto. Este comportamiento puede ser capturado de la siguiente manera:

```
umbral = 50

while timeRemaining(30):
  # Obtenemos los valores de los sensores de luz derecho e izquierdo
  L = getLight('left')
  R = getLight('right')

  # decidimos cómo actuar en base  a los valores de los sensores
  if (L - R) > umbral:
    # el izquierdo parece ser menor que el derecho
    turnRight(1.0)
  elif (R - L) > thresh:
    # el derecho parece ser menor que el izquierdo
    turnLeft(1.0)
  else:
    # la diferencia es menor que el umbral: quedarse quieto
    stop()
```

Observen cómo hemos usado la variable `umbral` para representar el valor del umbral. Esta es una buena práctica de programación. Dado que la performance de los sensores varía bajo diferentes condiciones de luz, esto les permite ajustar el umbral con simplemente cambiar ese valor. Usando el nombre `umbral` en lugar de un valor fijo, digamos 50, solamente deben hacer ese tipo de cambios en un lugar del programa.

En las sentencias de arriba hay un patrón que encontrarán recurriendo a muchos programas que definen comportamientos de robot usando decisiones simples.

```
while timeRemaing(<segundos>):
  <sensar>
  <decidir y actuar>
```

Tales comportamientos se denominan comportamientos *reactivos*. Es decir que un robot está reaccionando al cambio en su entorno tomando la decisión acerca de la manera de actuar basándose en los valores de su sensor. Un amplio rango de comportamientos de robot pueden ser escritos usando esta estructura de programa.

Abajo presentamos descripciones de varios comportamientos de robot automatizados interesantes, aunque simples. Siéntanse libres de implementar alguno de ellos en su robot.

Comportamientos Reactivos Simples

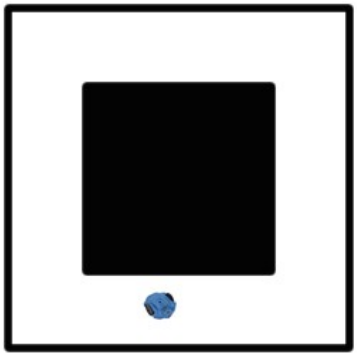
La mayoría de los comportamientos descriptos abajo requiere una selección entre alternativas usando expresiones condicionales y sentencias-if.

Heladera detective: Cuando eran niños ¿nunca se preguntaron si siempre estaba encendida la luz de la heladera? ¿O si se apagaba cuando cerraban la puerta? Bueno, aquí hay una forma de averiguarlo. ¡Construyan un robot detective que se siente dentro de la heladera y les diga si la luz está encendida o apagada!

Robot alarma contra robos: Diseñen un robot que cuide la puerta de su pieza. Ni bien se abra la puerta, que haga sonar una alarma (un beep).

Detector de paredes: Escriban un programa para el robot que avance derecho y se detenga cuando detecte una pared frente a él. Estarán usando los sensores IR para esta tarea.

Navegador de pasillo: Imaginen a su robot en un entorno que tiene un corredor con paredes que rodea a una caja cuadrada de 60cm. por 60cm. (ver la imagen al lado). Escriban un programa que le permita al robot avanzar alrededor de esa caja. Una estrategia que pueden usar es que el robot vaya hacia delante en una línea recta hasta que se choque contra una pared. Luego de chocar, realizará un giro de 90 grados (quizás necesiten retroceder un poco para permitir espacio para el giro) y luego continuará en línea recta.



Dispositivo de medición: Han calibrado al robot con respecto a cuánta distancia viaja en cierta cantidad de tiempo. Pueden usar esto para diseñar un robot que mida espacios. Escriban un programa que le permita al robot medir el ancho de un pasillo.

El perseguidor: Escriban un programa de robot que exhiba el siguiente comportamiento: El robot prefiere quedarse cerca de una pared (en frente). Si no tiene una pared enfrente, se mueve hacia delante hasta que la encuentra. Testen el programa en primer lugar ubicando al robot en un corralito de juego. Asegúrense de que el programa se comporte como en la descripción. A continuación, ubíquenlo en el piso y sostengan un papel blanco frente a él (lo suficientemente cerca como para que lo detecte). Ahora, lentamente, alejen el papel del robot. ¿Qué sucede?

Diseño de Comportamientos Reactivos

La mayoría de los comportamientos que se implementan usando el estilo Braitenberg dependen de algunas cosas simples: seleccionar uno o más sensores; elegir el tipo de cableado (directo o cruzado); y seleccionar las funciones de normalización para cada sensor. Mientras que pueden adivinar el comportamiento que puede resultar de estos tres diseños, la única forma de confirmarlo es directamente observando al robot llevar a cabo estos comportamientos. También han visto cómo, usando las sentencias-if pueden diseñar comportamientos simples, aunque interesantes. En esta sección diseñaremos comportamientos reactivos adicionales.

Seguimiento de la Luz

Para empezar, será bastante directo extender el comportamiento a partir de la orientación de luz desde arriba en otro que resulte en un robot seguidor de la luz. Es decir, con una linterna, podrán guiar al robot para que los siga. Nuevamente, tendrán que empezar por observar el rango de valores reportado por el robot bajo varias condiciones de luz. Si una linterna va a ser una fuente de luz brillante, observarán que los sensores de luz reportan muy bajos valores cuando una luz los ilumina directamente (típicamente en el rango de 0..50). Entonces, la decisión de para qué lado ir (hacia delante, doblar a la izquierda o hacia la derecha) puede tomarse basándose en las lecturas del sensor de los tres sensores de luz. La estructura del programa aparece de esta manera:

```
#Seguidor de luz

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa

umbral = 50
velocidadAdelante = 0.8
velocidadCrucero = 0.5
velocidadGiro = 0.7     # esto girará a la izquierda, -0.7 girará a la derecha

def main():
   while True:
      # obtener los valores de los sensores izquierdo, central y derecho
      L, C, R = getLight()

      # decidimos cómo actuamos en base a los valores de los sensores
      if C < umbral:
         # luz brillante desde el centro, ir derecho
         move(velocidadAdelante, 0)
      elif L < umbral:
         # luz brillantea la izquierda, girar a la izquierda
         move(velocidadCrucero, velocidadGiro)
      elif R < umbral:
         # luz brillantea la derecha, girar a la derecha
         move(velocidadCrucero, -velocidadGiro)
      else:
         # sin luz, moverse desapacio (¿o parar?)
         move(velocidadCrucero/2, 0)
main()
```

Noten que en el programa de arriba, hemos decidido establecer valores para el umbral de luz (`umbral`) así como movimientos para valores específicos. Esto se debe a que el comando `move` nos permite mezclar movimientos de traslación y de rotación. Adicionalmente, observen que más allá de los valores del sensor, el robot siempre se mueve hacia delante un poco aún cuando está doblando. Esto es esencial dado que el robot debe seguir la luz y no sólo orientarse hacia ella. En el caso en el cual no hay una luz brillante presente, el robot aún se mueve hacia delante (a mitad de la velocidad crucero).

Realizar la siguiente actividad: Implementar el programa de seguimiento de luz como se describe arriba y observen el comportamiento del robot. Intenten ajustar los valores establecidos (para umbral y para velocidades de motor) y noten los cambios en los comportamientos del robot. Además, ¿observan que este comportamiento es similar a alguno de los vehículos que se describen arriba? ¿Cuál?

En el diseño del robot seguidor de luz de arriba, usamos un valor de umbral para detectar la presencia de brillo de luz. A veces es más interesante usar umbrales diferenciales para valores de sensor. Es decir, ¿el valor del sensor de luz es diferente de la luz ambiente por cierta cantidad de umbral? Pueden volver a utilizar la función `senses` y observar las diferencias de la luz ambiental y modificar el programa de arriba para usar el diferencial en lugar del umbral fijado.

Aquí hay otra idea. Junten a varios de sus compañeros de clase en una habitación con sus robots, todos corriendo el mismo programa. Asegúrense de que a habitación tenga suficiente espacio en el piso y una gran ventana con cortina.

Cierren las cortinas para bloquear temporalmente la luz externa. Ubiquen a los robots en toda la habitación e inicien el programa. El robot se escurrirá por ahí, en dirección a su orientación inicial. Ahora lentamente abran las cortinas para que entre más luz. ¿Qué ocurre?

Esquivando Obstáculos

Los obstáculos en el camino del robot pueden ser detectados usando los sensores IR frente al robot. Entonces, basándose en los valores obtenidos, el robot puede decidir alejarse del obstáculo que se avecina usando el siguiente algoritmo:

```
if obstacle straight ahead, turn (left or right?)
if obstacle on left, turn right
if obstacle on right, turn left
otherwise cruise
```

Esto puede ser implementado usando el programa de abajo:

```
# Esquivando Obstáculos

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa ...

velocidadCrucero = 0.6
velocidadGiro = 0.5     # esto girará a la izquierda, -0.5 girará a la derecha

def main():
   while True:
      # obtener los valores de los sensores IR izquierdo y derechos
      L, R = getIR()
      L = 1 - L
      R = 1 - R

      # decidimos cómo actuamos en base a los valores de los sensores
      if L and R:
         # obstáculo a la vista, girar (aleatóriamente)
         move(0, velocidadGiro)
      elif L:
         # obstáculo a la izquierda, girar a la derecha
         move(velocidadCrucero, -velocidadGiro)
      elif R:
         # obstáculo a la derecha, girar a la izquierda
         move(velocidadCrucero, velocidadGiro)
      else:
         # sin obstáculos
         move(velocidadCrucero, 0)
main()
```

Como en el caso del perseguidor de luz, observen que empezamos estableciendo valores para los movimientos. Adicionalmente, hemos dado vuelta los valores de los sensores IR para que las condiciones de las sentencias-if parezcan más naturales. Recuerden que los sensores IR reportan un valor de 1 en ausencia de un obstáculo y un 0 en presencia de uno. Al invertirlos (usando valor –1), el valor para un obstáculo es 1 y en caso contrario, es 0. Estos valores hacen que resulte más natural escribir

las condiciones en el programa de arriba. Recuerden que en Python un 0 es equivalente a `False` (falso) y un 1 es equivalente a `True`. Lean el programa de arriba cuidadosamente y asegúrense de entender estas sutilezas. Más allá de esto, la estructura del programa es muy similar a la del programa del perseguidor de luz.

Otra forma de escribir un comportamiento de robot similar es usando el valor del sensor de atascamiento. Recuerden que el sensor de atascamiento detecta si el robot se ha chocado contra algo. Entonces, pueden escribir un comportamiento que no necesariamente evita obstáculos, pero que recorre el entorno chocando con las cosas. Es similar a una persona que entra en una habitación oscura e intenta sentir su avance tocando o chocándose lentamente contra las cosas. En el caso del robot, no hay forma de saber si el choque fue en su izquierda o derecha. Sin embargo, si usan el programa (mostrado abajo) observarán un comportamiento bastante robusto por parte del robot.

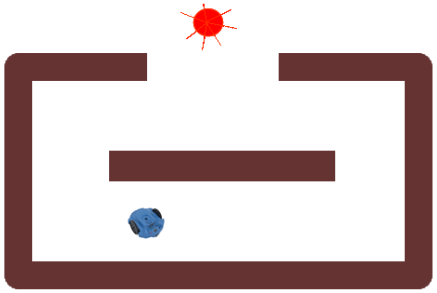
```
# Evitando obstáculos

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa ...

velocidadCrucero = 1.0
velocidadGiro = 0.5     # esto girará a la izquierda, -0.5 girará a la derecha

def main():
   while True:

      if getStall():
         # Estoy trabado, girar (aleatoriamente?)
         move(0, velocidadGiro)
      else:
         # No estoy trabado
          move(velocidadCrucero, 0)

main()
```

En ciertos momentos, notarán que el robot se atasca aún al intentar doblar. Una solución es detener al robot, retroceder un poco y luego doblar.

Realizar la siguiente actividad: Implementen el programa de arriba, observen el comportamiento del robot. A continuación, modifiquen el programa sugerido arriba (al quedar atascado, detenerse, retroceder y luego doblar).

Robot que resuelve laberintos: Creen un laberinto simple para el robot. Ubiquen al robot en una punta y usen los programas para esquivar obstáculos de arriba (ambas versiones). ¿El robot resuelve el laberinto? Si no lo hace, observen si el laberinto es para zurdos o derechos (por ej, todos los giros son para la derecha o para la izquierda), o ambos. Modifiquen los programas para esquivar obstáculos para resolver laberintos de derechos o de izquierdos. ¿Cómo le permitirían al robot solucionar un laberinto que tiene giros hacia la derecha y la izquierda?

¿Corral OK?



Salida del Corral

Dado que un simple programa para evitar obstáculos le permite a un robot resolver un laberinto sencillo, podemos también diseñar comportamientos más interesantes sobre esto. Imaginen un corral: un área cerrada con particiones tipo laberinto y una entrada, con una fuente de luz en la entrada (ver imagen a la derecha). Dada la posición del robot, ¿podemos diseñar un comportamiento que le permita al robot salir del corral?

Se puede diseñar una solución para el corral específico que se muestra aquí: que siga cualquier pared hasta que vea una luz brillante y luego pasar a buscador de luz. ¿Puede el Scribbler ser diseñado para seguir una pared? Recuerden que el Fluke tiene sensores de obstáculo izquierdo y derecho que apuntan a sus costados. Otro abordaje será combinar el comportamiento de esquivar obstáculos de arriba con el comportamiento de buscador de luz. Es decir, en ausencia de luz brillante, el robot se moverá alrededor del corral, evitando obstáculos, y cuando perciba una luz brillante, se dirigirá hacia ella. La parte difícil aquí será detectar que ha salido del corral y que necesita detenerse.

Resumen

Braitenberg utiliza ideas muy simples para pensar el modo en que están conectados los cerebros y cuerpos de humanos y animales. Por ejemplo, en los humanos, los nervios ópticos (además de otros) tienen conexiones cruzadas dentro del cerebro. Es decir que los nervios del ojo izquierdo se conectan con el lado derecho del cerebro y viceversa. De hecho, se cruzan y cierta información de cada lado también está representada en el mismo lado (es decir, hay conexiones directas, además de cruzadas). Sin embargo, aún es un enigma para los científicos en por qué es así y cuáles, si es que hay alguna, son las ventajas y desventajas de este esquema. De modo similar, si observamos el comportamiento de los Vehículos 2a y 2b, fácilmente se puede ver allí paralelos con el comportamiento de varios animales, como las moscas que se orientan hacia fuentes de luz o de calor. Simples comportamientos de robot nos puede proveer profundas comprensiones del comportamiento complejo: que la observación y el análisis de algo es una tarea cuesta arriba si no se conoce la estructura interna. Construyendo estructuras internas simples podemos arribar a comportamientos de apariencia complejos. Se ha visto que estos comportamientos de apariencia complejos también influencian comportamientos grupales en los insectos (ver la imagen de artículo en la página siguiente). Es decir, que robots que no se parecen en nada a los insectos y de tamaño parecido al del Scribbler pueden ser usados para influenciar comportamientos de insectos en muchas situaciones.

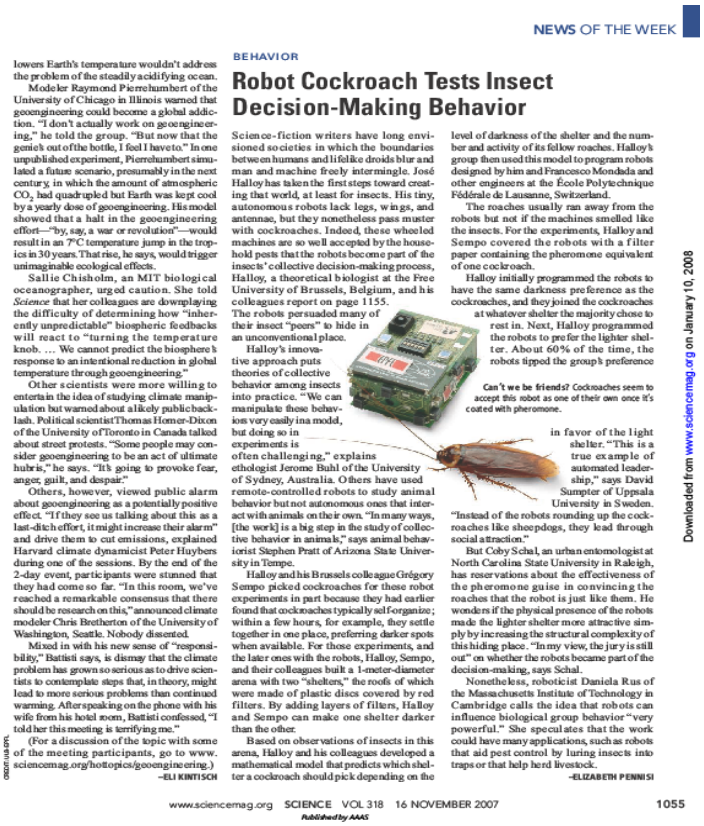
En este capítulo, hemos intentado darles un gustito de la idea de psicología sintética. A su vez, han aprendido cómo programar estructuras internas en un cerebro de robot y han aprendido varias técnicas para el control del robot.

Referencias

Todos los vehículos numerados descriptos aquí fueron desarrollados en un conjunto de experimentos pensados diseñados por Valentino Braitenberg en su libro, "*Vehicles: Experiments in Synthetic Psychology*" *(Vehículos: Experimentos en Psicología Sintética)*, MIT Press, 1984.

Algunos de los otros vehículos descriptos aquí fueron diseñados por David Hogg, Fred Martin, y Mitchel Resnickdel Laboratorio de Medios del MIT. Hogg et al usaron ladrillos LEGO especializados para construir estos vehículos. Para más detalles, ver su artículo titulado "*Braitenberg Creatures*" *(Criaturas de Britenberg)*.

Artículo de ***Science*** **Magazine, enero 10, 2008**



NEWS OF THE WEEK

BEHAVIOR

Robot Cockroach Tests Insect Decision-Making Behavior

lowers Earth's temperature wouldn't address the problem of the steadily acidifying ocean.

Modeler Raymond Pierrehumbert of the University of Chicago in Illinois warned that geoengineering could become a global addiction. "I don't actually work on geoengineering," he told the group. "But now that the genie's out of the bottle, I feel I have to." In one unpublished experiment, Pierrehumbert simulated a future scenario, presumably in the next century, in which the amount of atmospheric CO_2 had quadrupled but Earth was kept cool by a yearly dose of geoengineering. His model showed that a halt in the geoengineering effort—"by, say, a war or revolution"—would result in an 7°C temperature jump in the tropics in 30 years. That rise, he says, would trigger unimaginable ecological effects.

Sallie Chisholm, an MIT biological oceanographer, urged caution. She told *Science* that her colleagues are downplaying the difficulty of determining how "inherently unpredictable" biospheric feedbacks will react to "turning the temperature knob. … We cannot predict the biosphere's response to an intentional reduction in global temperature through geoengineering."

Other scientists were more willing to entertain the idea of studying climate manipulation but warned about a likely public backlash. Political scientist Thomas Homer-Dixon of the University of Toronto in Canada talked about street protests. "Some people may consider geoengineering to be an act of ultimate hubris," he says. "It's going to provoke fear, anger, guilt, and despair."

Others, however, viewed public alarm about geoengineering as a potentially positive effect. "If they see us talking about this as a last-ditch effort, it might increase their alarm" and drive them to cut emissions, explained Harvard climate dynamicist Peter Huybers during one of the sessions. By the end of the 2-day event, participants were stunned that they had come so far. "In this room, we've reached a remarkable consensus that there should be research on this," announced climate modeler Chris Bretherton of the University of Washington, Seattle. Nobody dissented.

Mixed in with his new sense of "responsibility," Battisti says, is dismay that the climate problem has grown so serious as to drive scientists to contemplate steps that, in theory, might lead to more serious problems than continued warming. After speaking on the phone with his wife from his hotel room, Battisti confessed, "I told her this meeting is terrifying me."

(For a discussion of the topic with some of the meeting participants, go to www.sciencemag.org/hottopics/geoengineering.)

–ELI KINTISCH

Science-fiction writers have long envisioned societies in which the boundaries between humans and lifelike droids blur and man and machine freely intermingle. José Halloy has taken the first steps toward creating that world, at least for insects. His tiny, autonomous robots lack legs, wings, and antennae, but they nonetheless pass muster with cockroaches. Indeed, these wheeled machines are so well accepted by the household pests that the robots become part of the insects' collective decision-making process, Halloy, a theoretical biologist at the Free University of Brussels, Belgium, and his colleagues report on page 1155.

The robots persuaded many of their insect "peers" to hide in an unconventional place.

Halloy's innovative approach puts theories of collective behavior among insects into practice. "We can manipulate these behaviors very easily in a model, but doing so in experiments is often challenging," explains ethologist Jerome Buhl of the University of Sydney, Australia. Others have used remote-controlled robots to study animal behavior but not autonomous ones that interact with animals on their own. "In many ways, [the work] is a big step in the study of collective behavior in animals," says animal behaviorist Stephen Pratt of Arizona State University in Tempe.

Halloy and his Brussels colleague Grégory Sempo picked cockroaches for these robot experiments in part because they had earlier found that cockroaches typically self-organize; within a few hours, for example, they settle together in one place, preferring darker spots when available. For those experiments, and the later ones with the robots, Halloy, Sempo, and their colleagues built a 1-meter-diameter arena with two "shelters," the roofs of which were made of plastic discs covered by red filters. By adding layers of filters, Halloy and Sempo can make one shelter darker than the other.

Based on observations of insects in this arena, Halloy and his colleagues developed a mathematical model that predicts which shelter a cockroach should pick depending on the level of darkness of the shelter and the number and activity of its fellow roaches. Halloy's group then used this model to program robots designed by him and Francesco Mondada and other engineers at the École Polytechnique Fédérale de Lausanne, Switzerland.

The roaches usually ran away from the robots but not if the machines smelled like the insects. For the experiments, Halloy and Sempo covered the robots with a filter paper containing the pheromone equivalent of one cockroach.

Halloy initially programmed the robots to have the same darkness preference as the cockroaches, and they joined the cockroaches at whatever shelter the majority chose to rest in. Next, Halloy programmed the robots to prefer the lighter shelter. About 60% of the time, the robots tipped the group's preference

Can't we be friends? Cockroaches seem to accept this robot as one of their own once it's coated with pheromone.

in favor of the light shelter. "This is a true example of automated leadership," says David Sumpter of Uppsala University in Sweden. "Instead of the robots rounding up the cockroaches like sheepdogs, they lead through social attraction."

But Coby Schal, an urban entomologist at North Carolina State University in Raleigh, has reservations about the effectiveness of the pheromone guise in convincing the roaches that the robot is just like them. He wonders if the physical presence of the robots made the lighter shelter more attractive simply by increasing the structural complexity of this hiding place. "In my view, the jury is still out" on whether the robots became part of the decision-making, says Schal.

Nonetheless, roboticist Daniela Rus of the Massachusetts Institute of Technology in Cambridge calls the idea that robots can influence biological group behavior "very powerful." She speculates that the work could have many applications, such as robots that aid pest control by luring insects into traps or that help herd livestock.

–ELIZABETH PENNISI

www.sciencemag.org SCIENCE VOL 318 16 NOVEMBER 2007 1055

Published by AAAS

Para leer más acerca de los robots influenciando el comportamiento de insectos, ver la revista "*Science*" de Noviembre 16, 2007. El artículo de base que se discute en la imagen de arriba es de Halloy *et al*, "*Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices*" *(Integración social de robots en grupos de cucarachas para controlar elecciones ato-organizadas), Science*, Noviembre 16, 2007. Volumen 318, pp 1155-1158.

Revisión Myro

No se introdujeron características nuevas de Myro en este capítulo.

Revisión Python

La sentencia-if en Python tiene las siguientes formas:

```
if <condicion>:
  <esto>

if <condicion>:
  <esto>
else:
  <eso>

if <condicion-1>:
  <esto>
elif <condicion-2>:
  <eso>
elif <condicion-3>:
  <otra cosa>
...
```

```
...
else:
   <otho>
```

Las condiciones pueden ser cualquier expresión que resulte en un valor True, False, 1, o 0. Revisen el Capítulo 4 para más detalles en la escritura de expresiones condicionales.

Ejercicios

1. Una forma aún mejor de promediar las condiciones de luz ambiente para su normalización es que el robot tome una muestra de la luz ambiente a su alrededor. Es decir, que dé un círculo completo y tome muestras de los diferentes valores del sensor de luz. El valor ambiente puede entonces establecerse como el promedio de todos los valores de luz. Escriban una función llamada `setAmbiente` que rote al robot en un círculo completo (o podrían usar el tiempo), y que tome muestras de los valores del sensor de luz a medida que rota, para finalmente devolver el promedio de todos los valores de luz. Modifiquen la línea:

```
Ambiente = sum(getLight())/3.0
```

con la línea:

```
Ambiente = setAmbiente()
```

Prueben todos los comportamientos anteriores descriptos en este capítulo para ver cómo este mecanismo nuevo afecta al comportamiento del robot.

2. Diseñen e implementen un programa que exhiba el comportamiento de salida del corral descrito en este capítulo.

3. Implementen en comportamiento de detective de heladera descrito en este capítulo.

4. Implementen el robot de Alarma contra robo descrito en este capítulo.

5. Implementen el comportamiento de navegador de pasillo descrito en este capítulo.

6. Además de movimientos, intenten integrar música/sonido a los comportamientos del robot y observen cómo el agregado de sonido amplifica la percepción de la personalidad del robot.