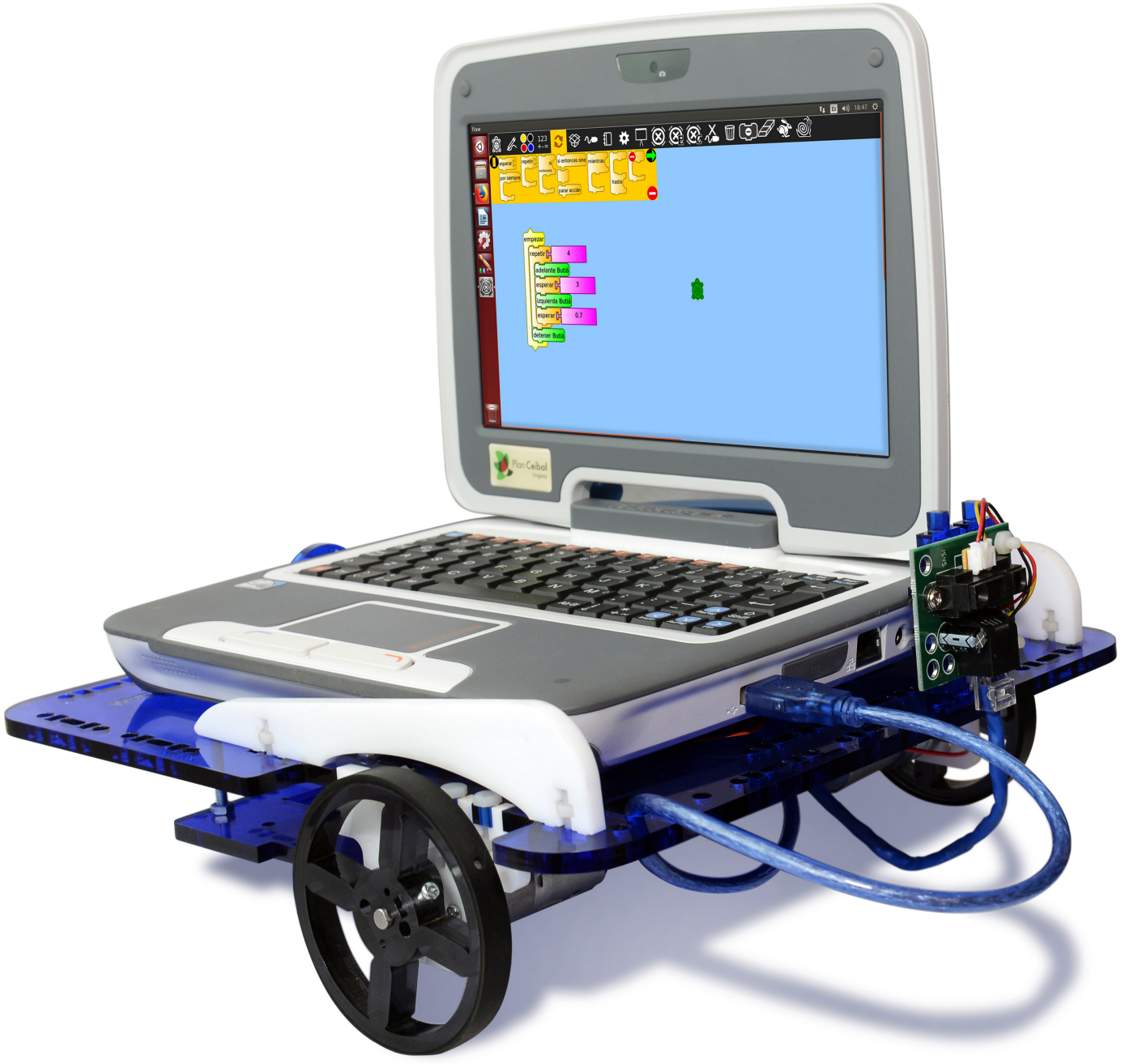
Manual de uso básico
de **TurtleBots** y robot **Butiá 2.0**

Alfabetización Digital, **Centros MEC**



Manual de uso básico de **TurtleBots** y robot **Butiá 2.0**

Alfabetización Digital, Centros MEC

Ministra de Educación y Cultura
Ma. Julia Muñoz

Subsecretaria de Educación y Cultura
Edith Moraes

Directora Centros MEC
Glenda Rondán

Sub Director Centros MEC
Fernando González

Jefa de área Alfabetización Digital y Proyectos Educativos
Ivonne Dos Santos

Autores
Bruno Michetti
Andrés Aguirre

Corrección
Facundo Bermudez
Silvana Tanca

Diseño gráfico y fotografía
Comunicación ministerial

Prólogo

La alfabetización digital es una propuesta que venimos desarrollando desde hace más de diez años en cada uno de los lugares donde funciona un Centro MEC.

En una primera instancia fueron talleres que empleaban diferentes metodologías los cuales, a partir de 2010, se transformaron en el Plan Nacional de Alfabetización Digital (PNAD), -el primero de su tipo en el país-, una política pública que busca complementar los esfuerzos del estado en materia de universalización del acceso y el uso de las tecnologías de la información y la comunicación (TICs).

En esta última década el mundo fue testigo de los cambios que se produjeron en el ámbito de las TICs y nuestro país no ha sido una isla en lo que respecta a esas transformaciones, algunas de las cuales son particularmente relevantes.

La aparición de nuevos soportes, aplicaciones, prestaciones, etcétera, ha colaborado con el objetivo de reducir la brecha digital, principal motor del PNAD, pero también ha vuelto más compleja la forma de abordar el tema.

Asimismo, el surgimiento de otros actores que se fueron desarrollando en el territorio -como por ejemplo del Plan Ibirapitá-, y la consolidación y reconocimiento innegable del Plan Ceibal, nos exige repensar lo que surgió como un plan de alcance nacional con fuerte vocación descentralizadora, en un momento en que nuestro estado distaba mucho de tener una presencia permanente a lo largo y ancho del país.

Justamente, como el Plan Nacional de Alfabetización Digital es un ejemplo de creación y transformación permanente -a la vez que un programa francamente descentralizador-, sus responsables directos siempre han mantenido una política de ojos y oídos atentos a las necesidades y demandas de las comunidades donde se desarrolla.

Esa actitud de ida y vuelta, de diálogo horizontal entre docentes y usuarios, enriqueció la herramienta, sumando nuevas propuestas a los componentes iniciales.

Así, en 2014, se realizó la primera formación en Robótica Educativa adaptada a docentes de Centros MEC de todo el país -instancia que se repitió en los años siguientes-, dotando de conocimientos a quienes se convertirían en formadores de formadores, reproduciendo los saberes adquiridos y captando a nuevos públicos ávidos de aprender y aprehender las nuevas tecnologías con el fin de aplicarlas a los desafíos cotidianos.

Pero esos cambios, la adaptación a la demanda que surge de la realidad a la que se enfrentan docentes y coordinadores de Centros MEC, así como la intención continua de transmitir los conocimientos de las TICs de la forma más democrática y universal o el desarrollo mismo del Plan Nacional de Alfabetización Digital, no nos desvían del objetivo principal: la formación de ciudadanía, el que todas y todos se apropien de sus derechos, los conozcan, los ejerzan y logren caminar hacia una vida más plena.

El PNAD es una herramienta en permanente transformación, pero cada modificación del mismo tiene presente que las personas son y serán las beneficiarias de nuestros desvelos.

Por eso, seguramente habrá una propuesta de alfabetización digital que se adapte a las necesidades de quienes se acerquen a Centros MEC.

Y si todavía no existe, la construiremos juntos.

Índice

Recurso educativo abierto 7

Objetivos del manual 8

1 TurtleBots 9
- 1.1 Instalación 10
  - 1.1.1 Conocer la versión de la computadora 10
  - 1.1.2 Descarga 10
  - 1.1.3 Comenzando la instalación 13
  - 1.1.4 Abriendo TurtleBots 13
- 1.2 Conectando el robot Butiá a la computadora 13
- 1.3 Paleta Butiá 15
- 1.4 Paleta de variables 17
- 1.5 Paleta de operadores numéricos 18
  - 1.5.1 Operadores aritméticos 19
  - 1.5.2 Operadores lógicos 21
- 1.6 Paleta de operadores de flujo 23
  - 1.6.1 Estructuras de control 24
  - 1.6.2 El bloque esperar 26
  - 1.6.3 Bloque espaciador 27
- 1.7 Borrar/restaurar bloques 28
- 1.8 Detener un programa que se está ejecutando 30
- 1.9 Guardar/cargar un programa 31
- 1.10 Debuggear un programa 32

2 Sensores del kit Butiá 34
- 2.1 Conectando sensores al robot 34
- 2.2 Testeo de sensores 36
- 2.3 Fichas de encastre y uniones 38
- 2.4 Sensor de contacto 39
- 2.5 Sensor de luz 40
- 2.6 Sensor de distancia 41
- 2.7 Sensor de escala de grises 42

3 Ejemplos de uso del robot Butiá 43
- 3.1 Cuadrado 43
  - 3.1.1 Desafío 43
  - 3.1.2 Solución 43
- 3.2 Seguidor de líneas 46
  - 3.2.1 Desafío 46
  - 3.2.2 Calibrando el sensor de escala de grises 46
  - 3.2.3 Pista ovalada 48
  - 3.2.4 Pista con forma genérica 50
- 3.3 Esquivador de obstáculos 52
  - 3.3.1 Desafío 52
  - 3.3.2 Solución 52
  - 3.3.3 Calibrando el sensor de distancia 53

Glosario 55

Recurso educativo abierto

Objetivos del manual

El presente *documento* es un manual que pretende dar una introducción al programa TurtleBots y explicar paso a paso diferentes utilidades del robot Butiá 2.0. Se espera que aquel educador que desee emplear el robot en sus aulas, pueda a través de este manual realizar los primeros pasos, para luego ir descubriendo un sinfín de usos nuevos. Este manual pretende ser una ayuda rápida y clara, para aquella persona que quiera realizar con autonomía actividades básicas usando el robot Butiá 2.0. Es importante destacar que para el presente documento se toma como punto de partida el manual de usuario 2.0 del Proyecto Butiá[1].

Nota: Es importante aclarar que para este manual se asume que la persona ya posee el robot Butiá armado y funcionando. No se explica cómo armarlo, en caso de tener el robot desarmado se puede acceder al manual de Armado de Butiá 2.0 [2].

1 Butiá 2.0, Manual de usuario. https://www.fing.edu.uy/inco/proyectos/butia/files/docs_butia2/Informe_cierre_del_proyecto_buti%C3%A12.0_rev1.pdf

2 Butiá 2.0, Manual de construcción. https://www.fing.edu.uy/inco/proyectos/butia/files/docs_butia2/manual_de_construccion_rev1.pdf

1 TurtleBots

TortugArte[3] es una actividad Sugar[4] inspirada en Logo[5] la cual pone al alcance de niños conceptos de programación, mediante una interfaz gráfica icónica, donde cada instrucción se presenta como un bloque. El proyecto Butiá realizó modificaciones sobre TortugArte agregando algunos plugins en forma de paletas que permiten controlar diferentes kits robóticos como: Butiá, Lego NXT, Lego WeDo, etc. Estas modificaciones dieron lugar a TurtleBots.

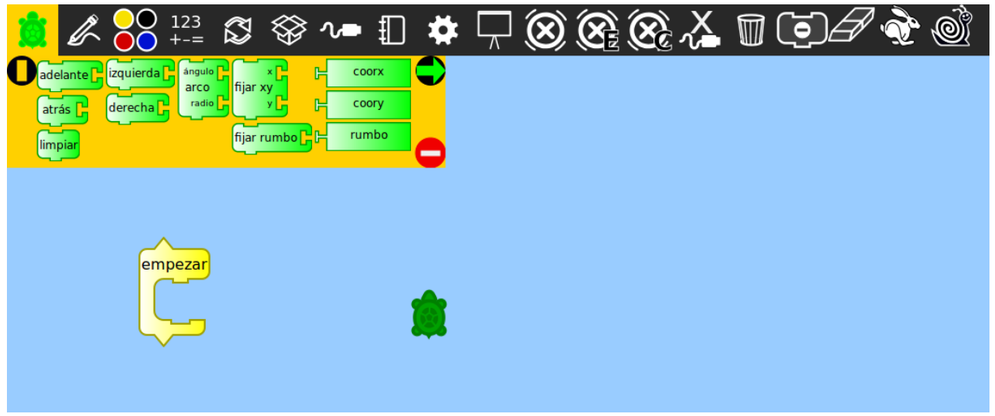


Figura 1.A: Programa TurtleBots

En TurtleBots escribir un programa significa encastrar bloques uno arriba del otro y colocarlos en el área celeste, dentro del bloque **empezar**. Para ejecutar un programa se tienen varios caminos igualmente válidos:

Hacer click arriba de empezar.
Hacer click arriba del primer bloque del programa.
Hacer click en el ícono del conejo:

Figura 1.B: Botón ejecutar

En el menú superior seleccionar **Tortuga > Ejecutar.**

3 TurtleArt. Logo Foundation. http://el.media.mit.edu/logo-foundation/services/ta.html
4 Sugar Labs. https://sugarlabs.org/
5 Logo Foundation. http://el.media.mit.edu/logo-foundation/

1.1 Instalación

En esta sección se muestra cómo instalar TurtleBots en la computadora. La instalación se debe hacer en una computadora que tenga sistema operativo GNU/Linux (la distribución más habitual es Ubuntu[6]). En la página del proyecto Butiá[7] se encuentra el instalador, sólo hay que ingresar a la página del proyecto, y luego dirigirse a descargas/Actividad TurtleBots. Nos aparecerán diferentes carpetas, debemos seleccionar la carpeta de la versión adecuada que dependerá de la versión de la computadora en la cual deseamos instalar el programa. Es importante aclarar que no debe usarse el browser Google Chrome para la descarga, dado que se han encontrado errores. El que se aconseja utilizar es Mozilla Firefox.

Nota: El tutorial paso a paso del presente documento se explica para computadoras con sistema operativo GNU/Linux de distribución Ubuntu, dado que es el sistema operativo que se encuentra en las computadoras Magallanes, y Positivo BGH.

1.1.1 Conocer la versión de la computadora

Por lo general la computadora puede tener una arquitectura de 32 bits (i386) ó de 64 bits (amd64). En esta sección se explica cómo saberlo fácilmente:

1. Abrimos la terminal, presionando:
   Ctrl + Alt + t
2. Escribimos en la terminal:
   `uname -m`
   Y presionamos enter.
3. Nos aparecerá un texto con el cual podemos ver la versión:
   a. Si aparece: `x86_64` entonces es de 64 bits (amd64).
   b. Si aparece: `i386` entonces es de 32 bits (i386).

1.1.2 Descarga

Una vez que sabemos la versión de la computadora en la que vamos a instalar TurtleBots, y hemos ingresado a la página indicada en la sección anterior, elegimos la carpeta correspondiente (i386 si es 32 bits, amd64 si es 64 bits). Luego seleccionamos el archivo con extensión *.deb* y número de versión de TurtleBots deseado.

6 Ubuntu Página oficial. https://www.ubuntu.com/
7 Proyecto Butiá. https://www.fing.edu.uy/inco/proyectos/butia/index.html

Nota: A efectos de este manual, no interesan los archivos de extensión *.changes* ni *.dsc*.
Actualmente existe hasta la versión 31 de TurtleBots, y se recomienda descargar cualquier versión entre la 29 y esta última. Lo mejor es descargar la versión 31 ya que es la que tiene más problemas solucionados, pero puede pasar que para computadoras que no sean muy nuevas se tenga algún problema de funcionamiento. Por lo que se puede proceder a descargar alguna versión anterior de TurtleBots. Como ejemplo, si la computadora es de 64 bits, se puede elegir para descargar alguno de estos instaladores:

- turtlebots_29_amd64.deb
- turtlebots_30_amd64.deb
- turtlebots_31_amd64.deb

1.1.3 Comenzando la instalación

Al finalizar la descarga tendremos un archivo con extensión *.deb*. El lugar del archivo dependerá de cada usuario. Por lo general se encuentra en la carpeta Descargas, pero puede tenerse configurado el navegador para seleccionar donde guardar lo que se descarga. Hay dos formas para realizar la instalación que se detallan a continuación.

1. La primer forma es hacer doble click sobre el archivo *.deb* descargado, y se abre entonces el *Centro de software de Ubuntu,* presionamos *Instalar*, ingresamos la contraseña, y clickeamos *Autenticar* para iniciar la instalación:

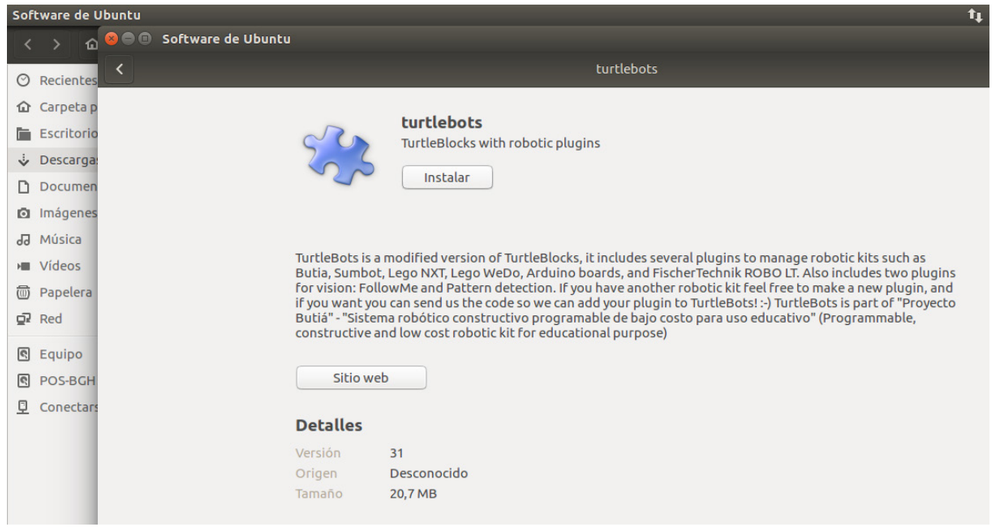


Figura 1.1.3.A: Instalación de TurtleBots paso 1

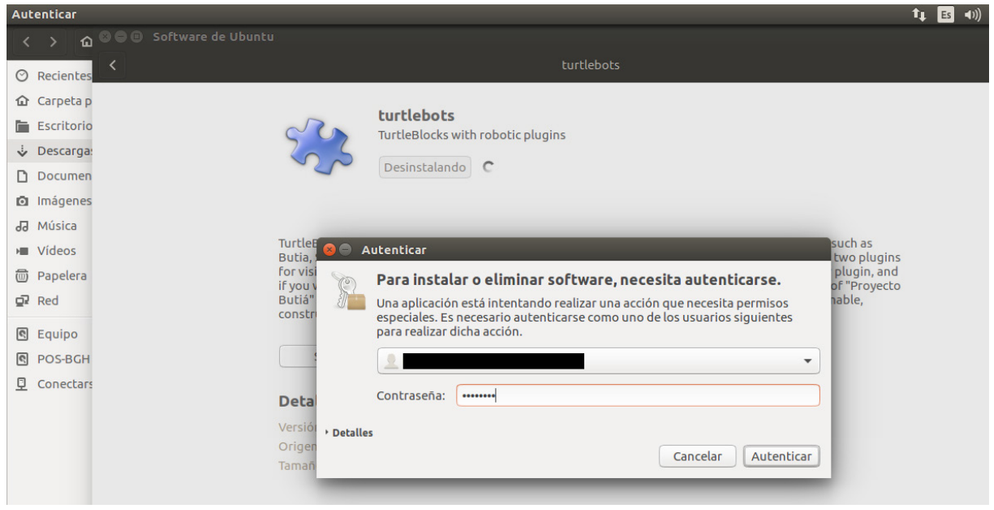


Figura 1.1.3.B: Instalación de TurtleBots paso 2

2. La segunda forma (ya que puede fallar a veces la primera) es hacerlo por línea de comandos:

a. Abrimos la terminal, presionando:
   Ctrl + Alt + t

b. Nos dirigimos a la carpeta en la cual se encuentra el archivo *.deb*, escribiendo en la terminal:
   `cd <ruta de la carpeta>`
   Y luego presionamos enter.

   Por ejemplo, si el archivo se encuentra en la carpeta *Descargas*, dentro de la carpeta *home*, ingresamos el siguiente texto en la terminal:
   `cd /home/Descargas` y presionamos enter.

c. Una vez parados en la carpeta que contiene al archivo *.deb*, escribimos:
   `sudo dpkg -i <nombre del archivo .deb>`
   Luego presionamos la tecla enter, e ingresamos la contraseña y comienza la instalación.

   Por ejemplo, si descargamos *turtlebots_30_amd64.deb*, ponemos:
   `sudo dpkg -i turtlebots_30_amd64.deb` y presionamos enter para comenzar la instalación.

Si no hubo problemas, hemos instalado TurtleBots correctamente.

Nota: Para saber la ruta de carpetas en la cual se encuentra un archivo, nos posicionamos sobre el archivo en cuestión, presionamos click derecho, vamos a *Propiedades* y se abre una ventana. En la misma vamos a la pestaña *Básico*, y vemos su ruta en el atributo *Lugar*.

1.1.4 Abriendo TurtleBots

Una vez que instalamos correctamente TurtleBots, podemos abrirlo de diferentes formas. A continuación se enumeran algunas, dependiendo de la computadora que se use:

I. Para cualquier computadora (Ubuntu, o plataforma Ceibal):
   A. Abrimos la terminal, presionando:
      Ctrl + Alt + t
   B. Escribimos:
      `turtlebots` y presionamos enter.

II. Para computadora con Ubuntu:
   A. Hacemos click en el botón de inicio, que se encuentra en el extremo superior izquierdo, y tiene el siguiente ícono:

Figura 1.1.4.A: Ícono Ubuntu

   B. Escribimos TurtleBots en la barra de búsqueda, y nos aparecerá el ícono del programa, al que le haremos click para abrir el programa.

III. Magallanes o Positivo BHG: En la barra de menú superior, nos dirigimos a **Aplicaciones > Educación > TurtleBots**.

1.2 Conectando el robot Butiá a la computadora

En este manual se asume que se tiene al robot Butiá 2.0 armado correctamente, por lo tanto se explicará solamente la conexión del mismo a la computadora (que a esta altura ya tiene instalado correctamente TurtleBots) y a los sensores (esto se verá en secciones posteriores).
El programa que controlará al robot Butiá se encuentra dentro de la computadora, por lo tanto es necesario conectarlo a la misma. Para ello utilizaremos el cable USB, que es el mismo cable que se utiliza en la mayoría de las impresoras:

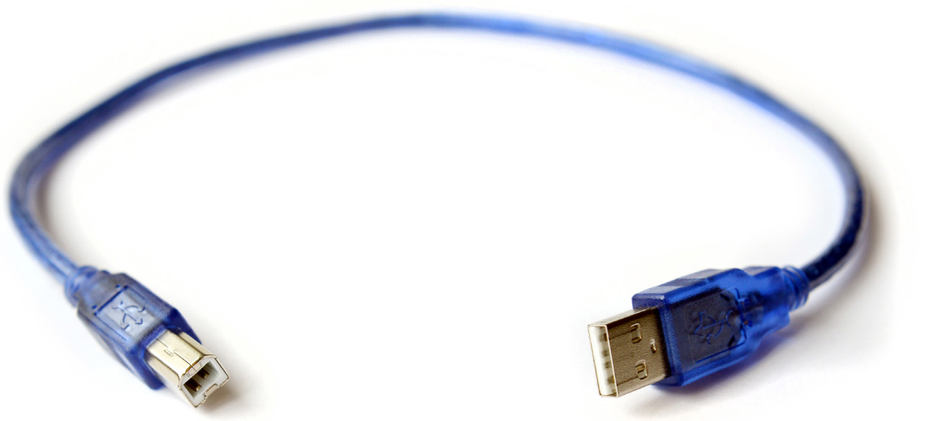


Figura 1.2.A: Cable USB

Se puede apreciar que el cable tiene un extremo que es USB, el mismo irá en cualquier puerto libre que tenga la computadora. El otro extremo irá en la placa USB4Butiá[8], que se encuentra en el chasis del robot, como se muestra en la siguiente foto:

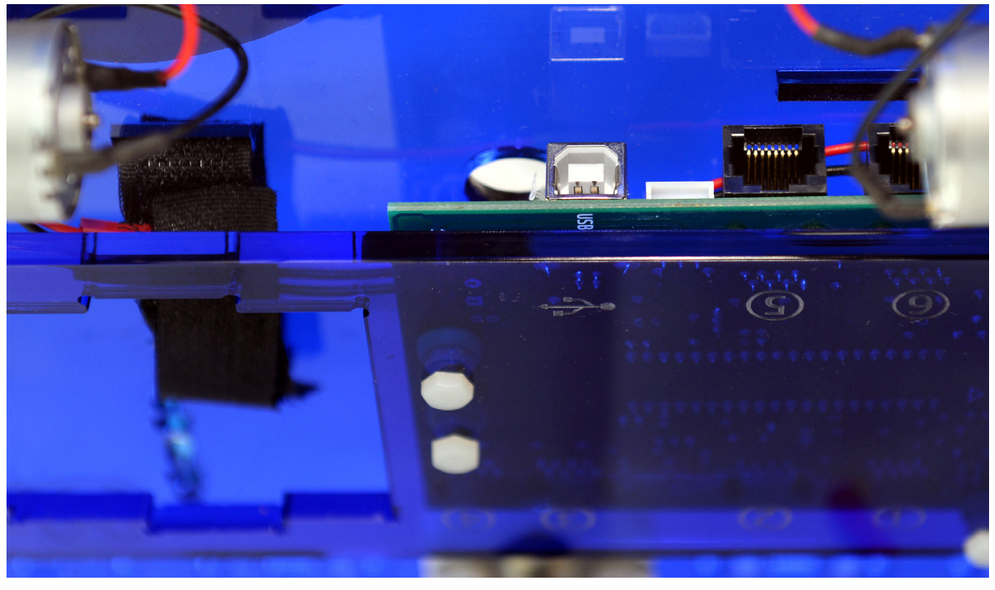


Figura 1.2.B: Entrada en placa USB4Butiá

8 USB4Butiá, Wiki Butiá. https://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/USB4buti%C3%A1

En el caso de que el cable sea muy largo y se pueda enredar con las ruedas u otras partes del robot, se puede enganchar en un hueco que hay en la placa del robot, como se muestra a continuación:

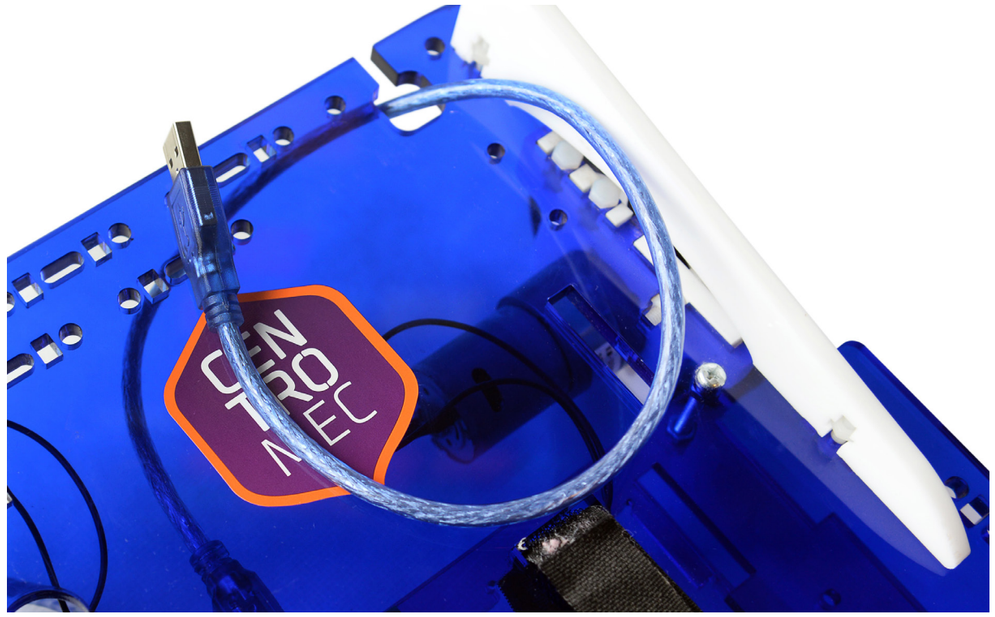


Figura 1.2.C: Hueco para cable USB

1.3 Paleta Butiá

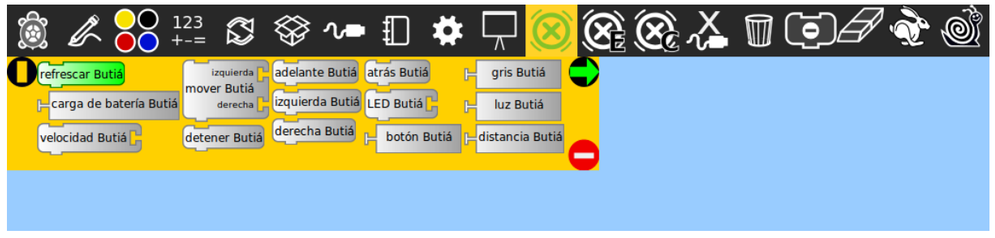


Figura 1.3.A: Paleta Butiá sin robot conectado

En TurtleBots las funcionalidades similares se agrupan en paletas que son los conjuntos de bloques de fondo amarillo, a los cuales se puede acceder presionando los botones que se encuentran en la barra superior de color negro. La paleta Butiá es una paleta desarrollada en la Facultad de Ingeniería, que contiene los bloques básicos que sirven para controlar al robot Butiá 2.0 y 1.0, y sus sensores básicos.

¿Por qué en la foto anterior se encuentran los bloques en color gris? Porque el robot Butiá tiene soporte *plug and play* y *hotplug* para los sensores/actuadores que se conectan a él, este aspecto fue utilizado a la hora de implementar el plugin cambiando el color de los bloques correspondientes a los elementos de sensado/actuación según lo que se haya conectado al robot[9].
Esto quiere decir que cuando conectemos el robot Butiá a la computadora, los bloques se volverán de color verde, indicando que se le pueden enviar esas órdenes al robot. En la foto a continuación, se muestra cómo quedan los bloques luego de conectar el Butiá 2.0 a la computadora.

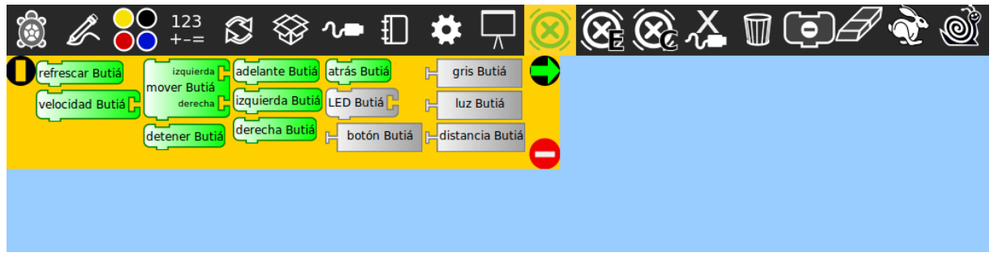


Figura 1.3.B: Paleta Butiá con robot conectado

¿Por qué luego de conectar el robot siguen apareciendo bloques en gris? Se explicará en breve. Por lo pronto se tienen en verde ya las órdenes básicas, éstas son:

- **adelante Butiá:** Con esta orden el robot gira sus motores para desplazarse hacia adelante.
- **atrás Butiá:** Análogo a la orden anterior pero hacia atrás.
- **izquierda Butiá:** El robot mueve sus motores de manera de girar sobre sí mismo hacia la izquierda, es decir en sentido antihorario.
- **derecha Butiá:** Análogo a la orden anterior pero hacia la derecha (sentido horario).
- **detener Butiá:** Con esta orden se detienen los motores.
- **velocidad Butiá:** Con este bloque se asigna la velocidad de ambos motores. Se usa un número entre 0 a 1023. Dichos números no son en metros por segundo, ni en ninguna magnitud de velocidad. Simplemente mientras mayor sea lo asignado, más potencia se envía a los motores para que se muevan más rápido.
- **mover Butiá:** Es análogo al bloque anterior, pero se asigna de forma independiente las velocidades para cada motor.

Los bloques restantes que siguen en gris, corresponden a los sensores del robot Butiá. Dichos bloques se ponen en verde cuando conectamos los diferentes sensores (que se verán en secciones posteriores).

Nota: En la versión 29 de TurtleBots es necesario presionar el botón **refrescar Butiá** luego de conectar el robot, para que los bloques se pongan en color verde.

9 Wiki Butiá, TurtleBots. https://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/TortuBots

1.4 Paleta de variables

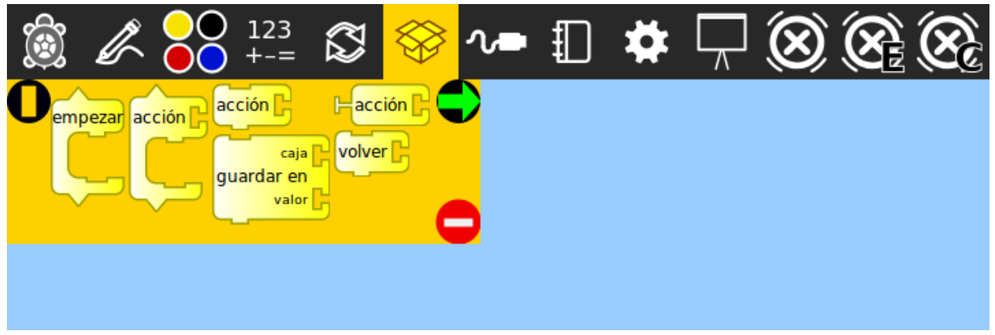


Figura 1.4.A: Paleta de variables

En esta sección se muestra cómo se trabaja con variables en TurtleBots. No se pretende realizar una introducción teórica al concepto de programación ni de variable, simplemente se explicará el funcionamiento y se darán ejemplos de uso.
Primero podemos ver el bloque **empezar**; se trata de la estructura en la cual se escribe el programa principal. Al abrir TurtleBots, aparece este bloque por defecto en el área celeste, donde "escribimos" los programas.
Las variables en TurtleBots se llaman cajas, y al igual que en cualquier otro lenguaje de programación las variables poseen un nombre; en el siguiente ejemplo podemos observar cómo guardar el valor 25 en una variable(caja) que llamamos "mi_variable":

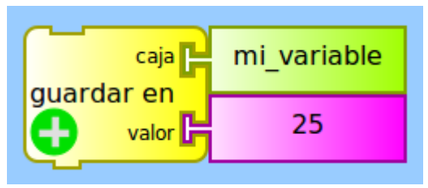


Figura 1.4.B: Ejemplo de definición de variable 1

Al ejecutar en un programa el bloque del ejemplo, suceden tres cosas:

1. Se almacena el valor 25 en la caja (la variable) de nombre "mi_variable".
2. Se crea en la paleta un nuevo bloque de la siguiente forma:

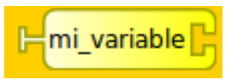


Figura 1.4.C: Ejemplo de definición de variable 2

3. Se crea en la paleta otro bloque de la siguiente forma:

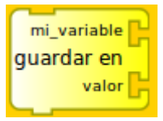


Figura 1.4.D: Ejemplo de definición de variable 3

El primer bloque creado en la paleta (Figura 1.4.C) sirve para consultar en el momento que se desee el valor que hay almacenado en la caja de nombre “mi_variable”. Si lo arrastramos al área celeste tendrá la siguiente forma:

Figura 1.4.E: Ejemplo de definición de variable 4

Al hacer click sobre éste último, podremos ver que se imprime el valor 25.0, que fue el que guardamos previamente. El segundo bloque creado en la paleta sirve para actualizar la variable cada vez que se desee. Si lo arrastramos al área celeste se verá igual que al comienzo. Tanto el nombre de la variable, como el valor son a modo de ejemplo.
En conclusión, en TurtleBots podemos almacenar, actualizar y consultar variables. Los bloques correspondientes a las acciones, no son indispensables y no se tratan en este manual.

1.5 Paleta de operadores numéricos

Figura 1.5.A: Paleta de operadores numéricos

La paleta de operadores numéricos, posee dos grupos de funcionalidades: uno para realizar operaciones aritméticas y componerlas, y otro para realizar operaciones lógicas y también poder componerlas. Si observamos la imagen detenidamente, podemos ver que algunos bloques tienen en su lado izquierdo una forma de “T acostada”, y otros una especie de semicírculo. Los primeros corresponden a las operaciones aritméticas, y los segundos a operaciones lógicas. Además los distintos tipos de bloques se pueden combinar entre sí, siempre y cuando puedan encastrarse.

1.5.1 Operadores aritméticos

Los operadores aritméticos corresponden a funciones que devuelven valores numéricos, por eso los bloques tienen en su lado izquierdo la forma de “T acostada”. Las operaciones que se pueden realizar son:

- Suma
- Resta
- Multiplicación
- División
- Módulo
- Raíz cuadrada

¿Qué es el bloque que tiene el número 100? Este bloque sirve para conectar a las operaciones que se enumeran arriba. El 100 es el valor que tiene por defecto, pero se puede hacer click sobre él, e ingresar cualquier valor que se desee. Estos valores pueden ser tanto enteros, como reales, y pueden ser positivos y negativos. Veamos algún ejemplo:

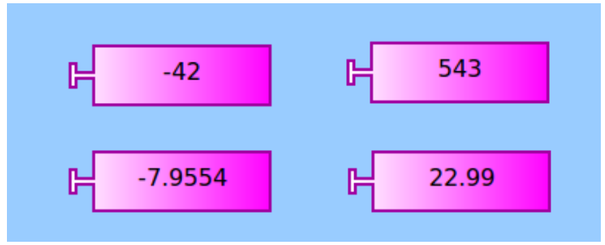


Figura 1.5.1.A: Ejemplo de números

El bloque que dice **aleatorio** recibe dos parámetros numéricos. Al ejecutar dicho bloque, se devuelve un número aleatorio entre el valor que se coloque en **min** y el que se coloque en **max**. Por ejemplo, si queremos un valor aleatorio entre 11 y 73, creamos el siguiente bloque:

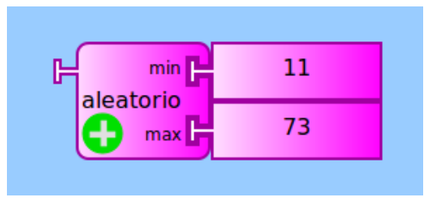


Figura 1.5.1.B: Ejemplo de bloque de número aleatorio

¿Qué significa un número aleatorio? Significa que al hacer click al bloque del ejemplo, la computadora generará al azar y devolverá un número que será mayor o igual a 11, y menor o igual a 73. Por lo tanto si le hacemos click varias veces al bloque, nos aparecerán números al azar dentro de ese rango. Su utilidad dependerá del problema que se quiera atacar.
Como se comentó anteriormente, se pueden combinar operaciones aritméticas. Por ejemplo se puede hacer 6 + 7 - 2, combinando el bloque de suma y de resta, de la siguiente manera:

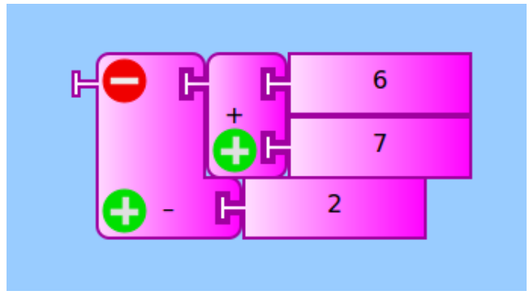


Figura 1.5.1.C: Ejemplo de cuenta combinada 1

Podemos armar más operaciones combinadas un poco más complejas, por ejemplo:
(-9 + 11)x(3 / 4):

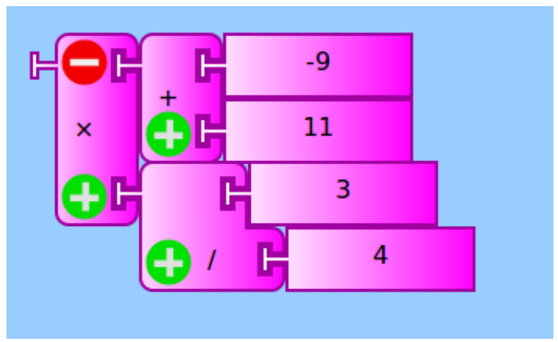


Figura 1.5.1.D: Ejemplo de cuenta combinada 2

Para ejecutar la operación combinada y obtener su resultado basta con hacer click sobre el bloque más a la izquierda. Podemos concluir que los bloques que tengan a su derecha la forma de "T acostada" reciben números para procesarlos, y los bloques que tengan a su izquierda la forma mencionada devuelven números.

Nota: Para escribir un número con coma, se puede hacer tanto con "." como con ",". Sin embargo una vez que hacemos click afuera se muestra el punto, como apreciamos en los ejemplos.

1.5.2 Operadores lógicos

Los operadores lógicos que tienen en su lado izquierdo la forma curva, devuelven sólo dos valores posibles: True (Verdadero) o False (Falso). Los bloques de "y","no" y "o" corresponden a las funciones lógicas AND, NOT y OR respectivamente. A continuación se muestran las tablas de entradas y salidas de las funciones mencionadas:

A	B	A AND B
False	False	False
True	False	False
False	True	False
True	True	True

A	NOT A
False	True
True	False

A	B	A OR B
False	False	False
True	False	True
False	True	True
True	True	True

Luego podemos ver los bloques con "<",">" y "=". Los mismos son también parte de los operadores lógicos, pero si miramos su forma apreciamos que del lado derecho tienen la ya mencionada "T acostada"; y esto es porque si bien devuelven resultados de True o False, reciben valores numéricos. A continuación se muestran ejemplos de combinación de bloques de la paleta de la presente sección:

- 3 es menor que 6

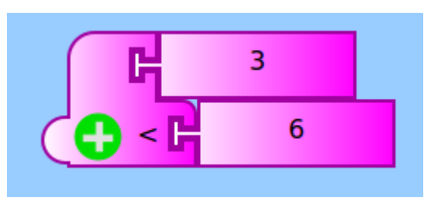


Figura 1.5.2.A: Ejemplo de expresión lógica 1

- 17 es igual a 11

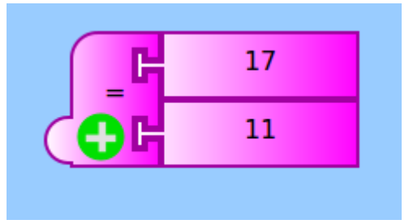


Figura 1.5.2.B: Ejemplo de expresión lógica 2

- 3,3 + 4 es mayor que 11

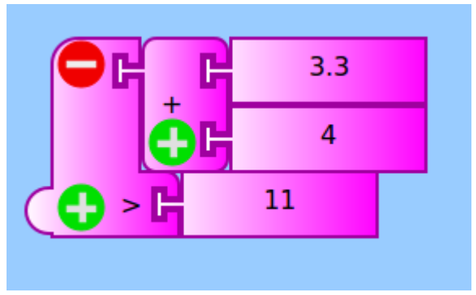


Figura 1.5.2.C: Ejemplo de expresión lógica 3

- 27 no es igual a 56

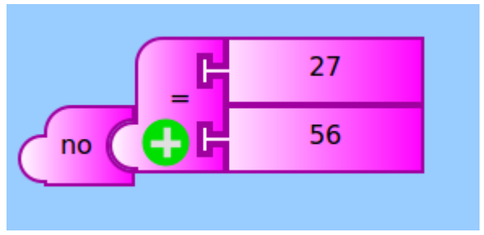


Figura 1.5.2.D: Ejemplo de expresión lógica 4

- 9 es menor que 11 y 24 es mayor que -5

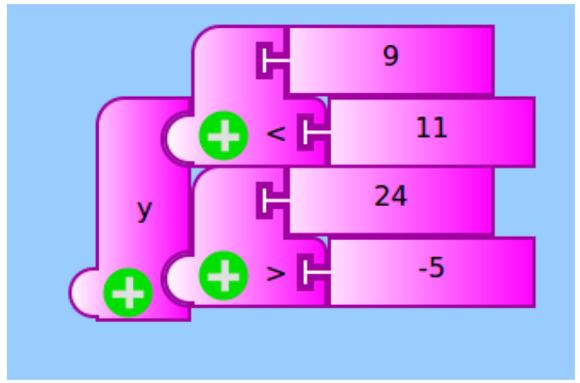


Figura 1.5.2.E: Ejemplo de expresión lógica 5

Para cada ejemplo mencionado se puede evaluar si es una expresión verdadera o falsa, si hacemos click sobre los bloques formados, se imprimirá en pantalla Verdadero o Falso según el caso. Más adelante en el manual veremos ejemplos de combinación de bloques entre distintas paletas.

1.6 Paleta de operadores de flujo

Figura 1.6.A: Paleta de operadores de flujo

En esta sección se muestra la paleta de operadores de flujo, que corresponden mayormente a estructuras de control en programación. En este manual no se pretende dar una introducción teórica de estructuras de control de programación, sino que se muestra la sintaxis de TurtleBots. La fundamentación teórica de los elementos de programación escapan a los objetivos del documento.

1.6.1 Estructuras de control

Las estructuras de control en TurtleBots son las siguientes:

- **si entonces:** Corresponde a la estructura **if-then**.
- **si entonces sino**: Corresponde a la estructura **if-then-else**.
- **repetir:** Corresponde a la estructura **for**.
- **mientras:** Corresponde a la estructura **while**.
- **por siempre:** Corresponde a realizar **while(true)**.
- **hasta:** Corresponde a la estructura **repeat-until**.

Si observamos las formas de los bloques que corresponden a estructuras de control que dependen de condiciones lógicas (Mientras, Si entonces, Hasta, etc), podemos ver que tienen un "hueco curvo". Ahí se deben colocar las condiciones lógicas que se pueden elaborar con la paleta de operadores numéricos. A continuación se ven ejemplos de programas en lenguaje TurtleBots, con sus respectivos pseudocódigos:

Pseudocódigo de Programa1:

Asignar a la variable **a** el valor **12**
Repetir **50** veces:
    Asignar a la variable **a** el valor que tiene, más **1**
Imprimir en pantalla el contenido de **a**

Código de Programa1 en TurteBots:

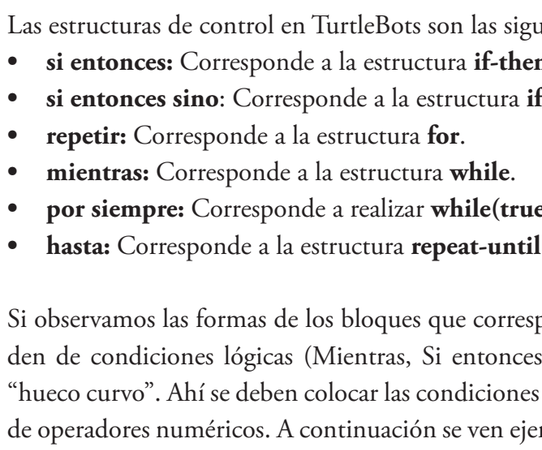


Figura 1.6.1.A: Ejemplo de programa 1

Pseudocódigo de Programa2:

Asignar a la variable **b** el valor **-2**
Mientras **b** sea menor a **51**:
    Imprimir en pantalla el contenido de **b**
    Asignar a la variable **b** el valor que tiene, más **3**

Código de Programa2 en TurtleBots:

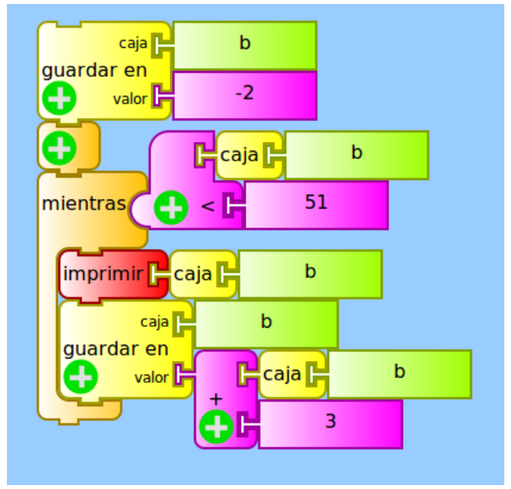


Figura 1.6.1.B: Ejemplo de programa 2

Es importante aclarar que el bloque imprimir se encuentra en la paleta de **opciones adicionales** que tiene el ícono:

Figura 1.6.1.C: Ícono de paleta de opciones adicionales

Su función es imprimir en pantalla el valor que recibe como parámetro. Si bien no se trata la paleta de opciones extra en el manual, dicho bloque es importante conocerlo y saber cómo funciona.

1.6.2 El bloque esperar

Este bloque no es una estructura de control, corresponde a una orden que permite detener la ejecución de nuestro programa durante un tiempo deseado. La computadora ejecuta instrucciones a una velocidad tan alta que si escribimos y ejecutamos el siguiente programa:

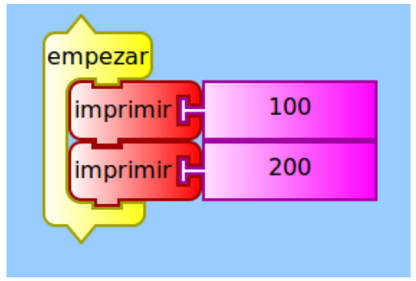


Figura 1.6.2.A: Ejemplo sin bloque esperar

Sólo veríamos el valor 200. Esto se debe a que si bien se ejecutó la instrucción de imprimir el valor 100, inmediatamente a una velocidad altísima imperceptible para el ojo humano se ejecutó la instrucción de imprimir 200.
A continuación se muestra un ejemplo en el que se imprime el valor 100, y luego de 3 segundos se imprime el valor 200.

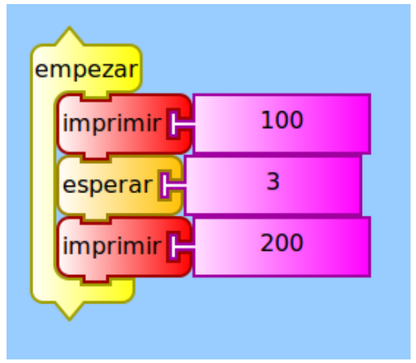


Figura 1.6.2.B: Ejemplo con bloque esperar 1

El bloque esperar sirve mucho cuando le mandamos instrucciones al robot Butiá. Esto se debe a lo siguiente: al ejecutar por ejemplo la instrucción **adelante Butiá**, se le envía a los motores la orden de girar para que el robot se mueva hacia adelante; y hasta que no se envíen nuevas órdenes a los motores, estos girarán hasta que el robot quede sin batería o sea desconectado. Por lo tanto con el bloque esperar podemos definir durante cuánto tiempo puede ejecutarse una orden hacia los motores. En el ejemplo a continuación se define un programa donde el robot avanza durante 3 segundos, luego gira sobre sí mismo hacia la izquierda durante 2 segundos, y luego se detiene.

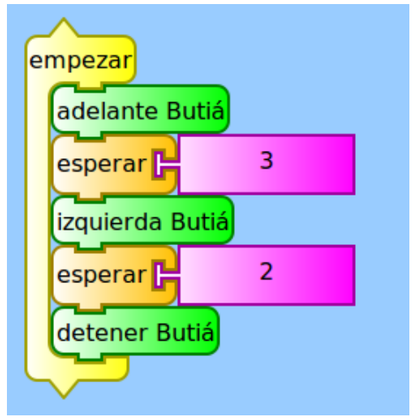


Figura 1.6.2.C: Ejemplo con bloque esperar 2

1.6.3 Bloque espaciador

Si observamos la paleta de operadores de flujo, u observamos el segundo programa de ejemplo (Figura 1.6.1.B), vamos a ver un bloque que no tiene nada escrito:

Figura 1.6.3.A: Bloque espaciador

¿Para qué sirve? Este bloque no tiene otra finalidad que hacer el código más legible. Veamos el siguiente ejemplo:

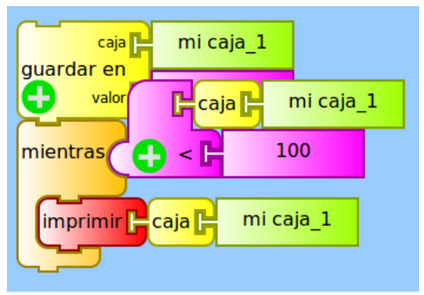


Figura 1.6.3.B: Ejemplo sin bloque espaciador

En este pequeño programa no podemos ver qué valor se guardará en la variable de nombre “mi_caja_1”. En vez de tener que desencastrar los bloques para ver bien, podemos usar el bloque espaciador de manera de que sea más fácil leer nuestro programa. Al arrastrar dicho bloque a la pantalla celeste, aparecerá un símbolo de “+” dentro del mismo, que nada tiene que ver con la suma aritmética. Si clickeamos ese símbolo el bloque empezará a estirarse hacia abajo y podemos ubicarlo en cualquier parte de nuestro programa para poder visualizar cosas que podrían quedar tapadas. Veamos cómo queda el código usando un bloque espaciador:

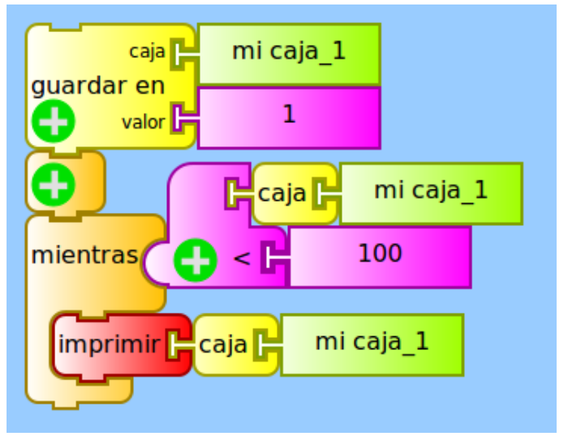


Figura 1.6.3.C: Ejemplo con bloque espaciador

Nota: Si en el bloque espaciador aparece un símbolo de “-” significa que puede achicarse. Podemos cambiar el tamaño de este bloque ajustándolo para espaciar nuestro código de una forma prolija.

1.7 Borrar/restaurar bloques

A veces sucede que luego de programar y realizar varias pruebas, la pantalla de TurtleBots se encuentra saturada de bloques en el área celeste. Esto puede ser molesto e incómodo para trabajar y resulta útil poder borrar aquellos bloques que no se usan. La forma de borrarlos es simplemente pinchar aquellos bloques que no usamos y deseamos quitar del área celeste, los arrastramos hasta la parte amarilla de la paleta (no interesa cuál es la paleta) y los soltamos. Al soltarlos se borran y de esta forma tenemos la pantalla más limpia para continuar nuestro trabajo.

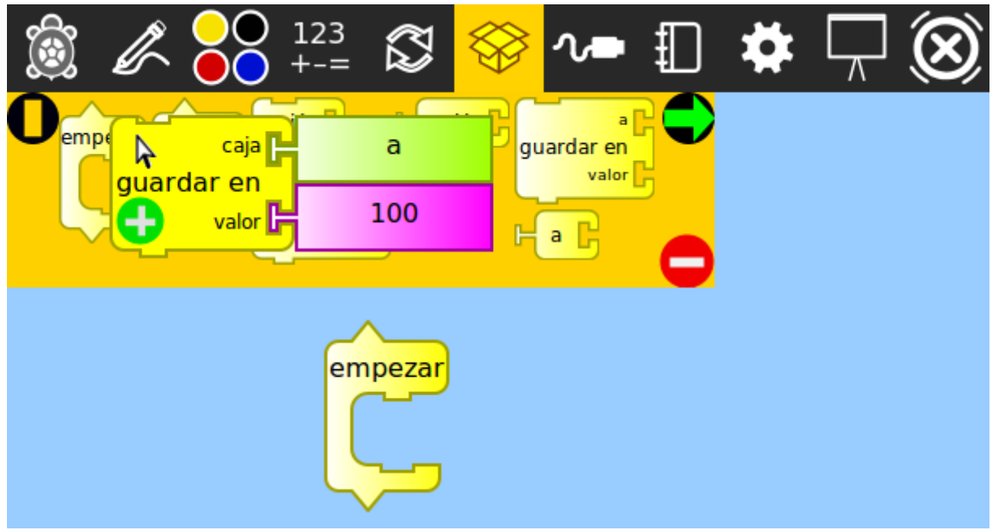


Figura 1.7.A: Ejemplo de borrado de un bloque

También se pueden borrar un conjunto de bloques encastrados, por ejemplo si tenemos un programa que ya no usamos y queremos hacer espacio en el área celeste, basta con arrastrar el primer bloque del conjunto, y soltarlo sobre la paleta (con el resto de los bloques enganchados debajo):

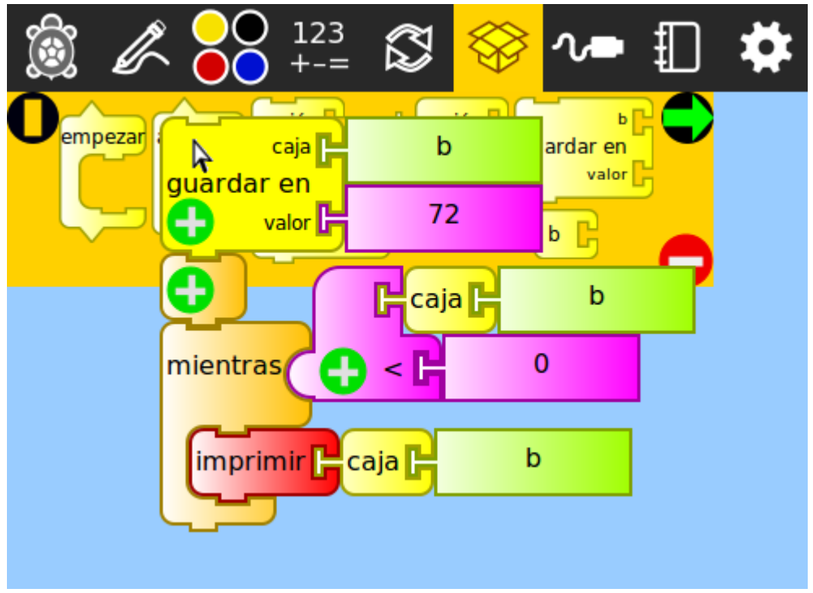


Figura 1.7.B: Ejemplo de borrado de varios bloques conectados

¿Es posible borrar todos los bloques (encastrados o no) del área celeste al mismo tiempo? Sí lo es, para ello nos dirigimos a la papelera de TurtleBots, que tiene el ícono:

Figura 1.7.C: Ícono de papelera

Luego seleccionamos **borrar todos** e inmediatamente se borran todos los bloques del área celeste. Ya que estamos en la papelera, podemos ver en ella los pequeños bloques de todo lo que hemos eliminado:

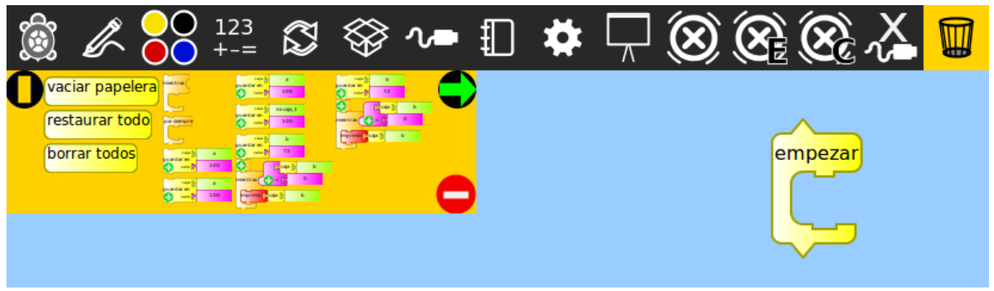


Figura 1.7.D: Ejemplo de bloques en la papelera

Puede suceder también que se hayan borrado bloques por error y deseamos recuperarlos, o simplemente queremos reutilizarlos. Para restaurarlos debemos hacer click sobre el bloque o conjunto de bloques eliminado/s que queremos recuperar. Al seleccionarlos aparecen nuevamente en el área celeste.
También podemos seleccionar **vaciar papelera** que elimina lo que se encuentra en ella y ya no puede ser recuperado, o **restaurar todo**, que vuelve todos los elementos eliminados al área celeste.

1.8 Detener un programa que se está ejecutando

Como vimos en la primera sección, hay varias formas de comenzar la ejecución de un programa. Mientras el mismo se ejecuta, puede suceder que el programa se quede trancado a causa de un error en la programación, o puede suceder que el robot esté haciendo algo que no se desea, y se torna necesario abortar dicha ejecución. Por lo tanto veremos algunas alternativas para salir de ese problema:

1. Presionar:
   Alt + s

2. Hacer click en el botón de detener programa:

Figura 1.8.A: Botón de detener programa

3. Hacer click en el menú de la barra superior en **Tortuga > Parar**.

1.9 Guardar/cargar un programa

Una vez que tenemos un programa y deseamos guardarlo, sólo debemos hacer click en el menú de la barra superior en **Archivo > Guardar**. Se abre una ventana que nos muestra nuestro sistema de archivos, y debemos seleccionar la carpeta donde queremos guardar el mismo.

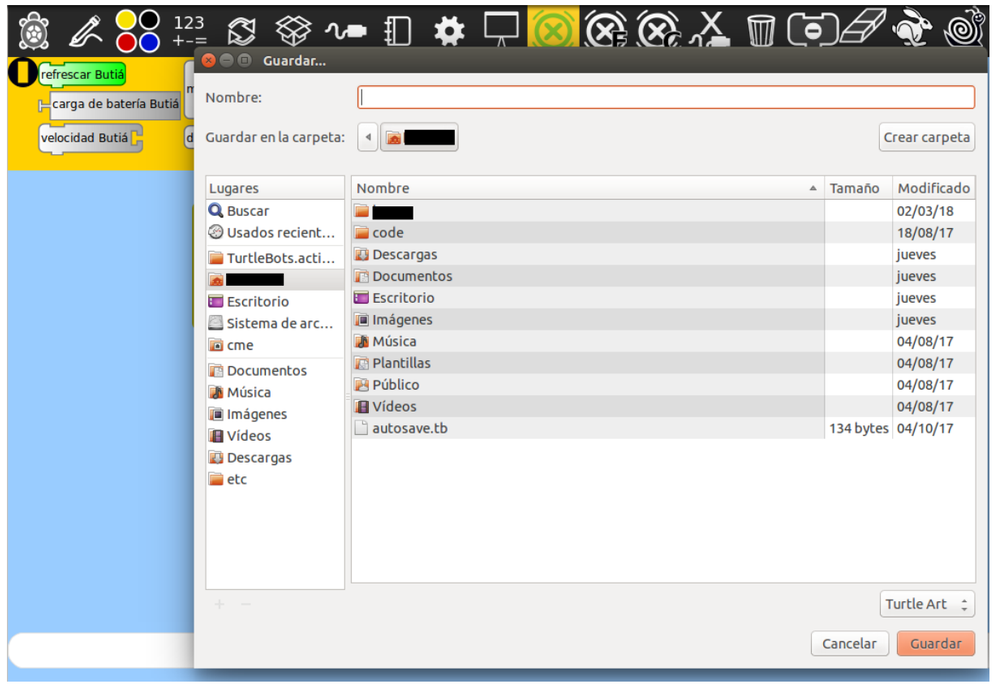


Figura 1.9.A: Ejemplo de guardar un programa

Luego de seleccionar la ubicación, nos dirigimos a la barra superior, y escribimos el nombre con el cual queremos guardar nuestro programa, por ejemplo “programa_prueba”. Luego hacemos click en el botón **Guardar**. A partir de este momento tendremos un archivo de nombre **programa_prueba** y extensión **.tb** guardado en la carpeta previamente seleccionada.
Si queremos abrir un programa, hacemos click en el menú de la barra superior en **Archivo > Abrir**. Ahora, se abre una ventana con nuestro sistema de archivos y nos dirigimos a la carpeta en donde tenemos nuestro archivo **.tb** guardado. Lo seleccionamos y clickeamos el botón de **Abrir**. En este momento aparecerá el programa tal cual lo guardamos la última vez.

1.10 Debuggear un programa

A veces nos sucede que el programa tiene errores, pero no sabemos por qué, sobre todo al programar robots. Debuggear un programa es buscar en él las posibles causas de los errores, por lo tanto TurtleBots nos permite ejecutar cada programa “más despacio” que lo normal y además nos muestra el color acentuado del bloque que se está ejecutando en cada momento.
En el siguiente ejemplo vemos que se está debuggeando un programa y en el instante de la foto se está ejecutando la instrucción de **esperar** (la que se encuentra inmediatamente después de ejecutar **adelante Butiá**).

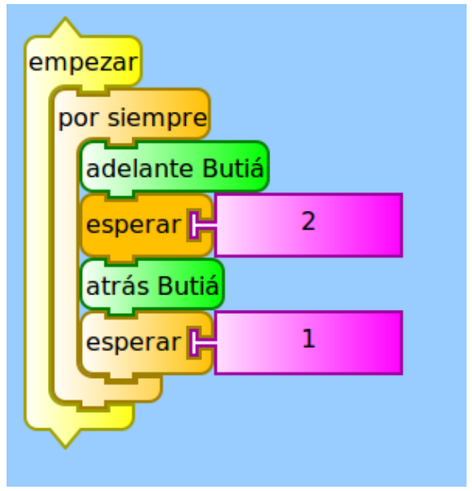


Figura 1.10.A: Ejemplo de debuggear un programa

De esta forma podremos ver en cada momento qué bloque se está ejecutando, y a la misma vez observamos al robot, por lo tanto nos será más sencillo encontrar errores en el programa construído. Para debuggear un programa, es decir ejecutarlo más lentamente y ver el color más intenso del bloque actual, tenemos dos formas:

1. Hacer click en el botón

Figura 1.10.B: Botón de ejecutar despacio un programa

2. En el menú superior seleccionar **Tortuga > Depurar**.

Nota1: Existen otras formas de debuggear un programa, pero no se tratan en este manual.
Nota2: Las paletas y funcionalidades mencionadas en las subsecciones anteriores no son para uso exclusivo del robot Butiá (excepto la paleta Butiá), son de uso general de TurtleBots.

2 Sensores del kit Butiá

El robot Butiá dispone de un conjunto de sensores básicos, sencillos de utilizar y diseñados para que se puedan construir por los usuarios con materiales baratos. Si bien escapa a los objetivos de este manual, es importante aclarar que el robot Butiá 2.0 fue diseñado para que pueda ser construído por usuarios y con materiales de bajo costo[10]. A grandes rasgos, los sensores son los elementos del robot que sirven para brindarle información sobre el entorno en el que se encuentra, de forma análoga a los sentidos para una persona. En esta sección se enumeran los sensores y una guía de cómo calibrarlos y usarlos.

Nota: En esta sección solo se hablará de los sensores del kit básico de Butiá, pero se pueden encontrar en la wiki butiá otros sensores 2.0[11] por ejemplo el sensor voltaje, o de resistencia.

2.1 Conectando sensores al robot

Se puede observar que en la placa USB4Butiá, se dispone de 6 puertos RJ45 (los puertos idénticos a los que conectamos internet cableado a una pc), que están enumerados del 1 al 6 en el chasis, como apreciamos en la siguiente imagen:

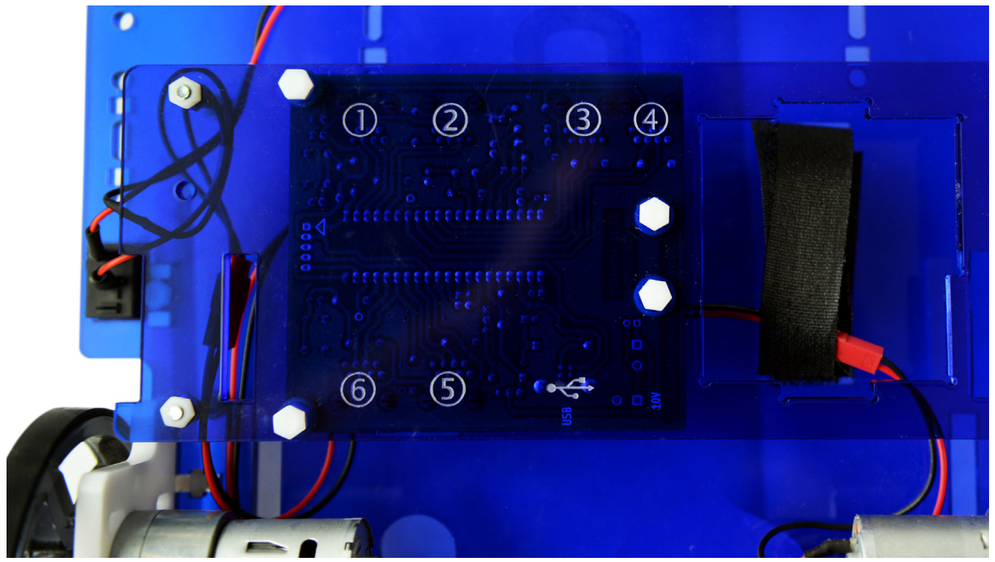


Figura 2.1.A: Chasis y puertos del robot Butiá

10 Butiá 2.0, Manual de construcción. https://www.fing.edu.uy/inco/proyectos/butia/files/docs_butia2/manual_de_construccion_rev1.pdf

11 USB4Butiá, Wiki Butiá. https://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/USB4buti%C3%A1

En estos puertos podemos conectarle entonces hasta 6 sensores (cualquiera del kit) de forma simultánea. Para ello se tienen los cables de sensores, que tienen en ambas puntas fichas RJ45:

Figura 2.1.B: Cable de sensor

Un extremo del cable lo conectamos a un sensor, y el otro lo conectamos a cualquiera de los puertos de la USB4Butiá, bajo el chasis del robot:

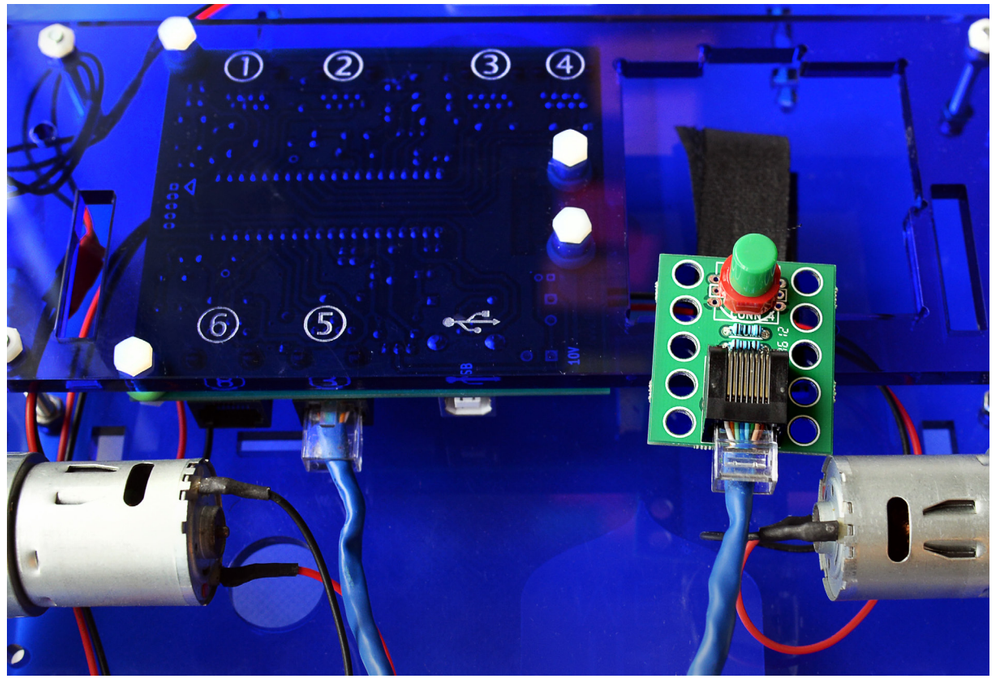


Figura 2.1.C: Sensor conectado al robot

¿Qué pasa en TurtleBots cuando conectamos sensores? Si vamos a la paleta Butiá, los bloques en gris correspondientes a los sensores conectados, se pondrán de color verde, indicando que ahora se pueden utilizar los valores que ellos devuelven. En el ejemplo a continuación se muestra la paleta Butiá luego de haber conectado un sensor de distancia y dos sensores de grises:

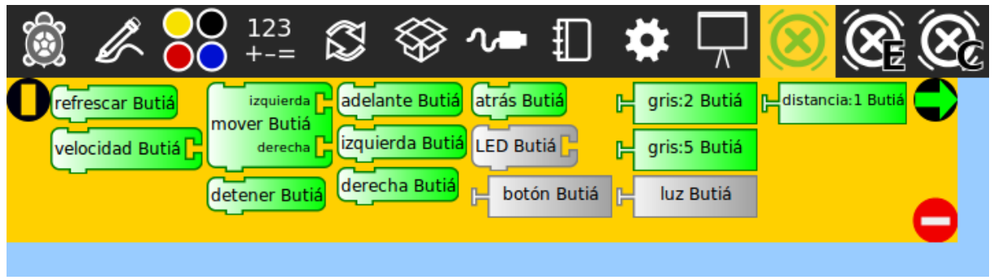


Figura 2.2.D: Paleta Butiá con sensores conectados

¿Qué significan los números en los bloques de los sensores? Una vez que TurtleBots reconoce que se ha conectado un sensor al robot Butiá, reconoce automáticamente en qué puerto fue conectado. Por lo tanto el 1 indica que se conectó el sensor de distancia en el puerto 1, y el 2 y 5 significa que se tienen los sensores de escala de grises en los puertos 2 y 5. Esto es muy útil si conectamos dos sensores del mismo tipo, y de alguna forma queremos distinguirlos para programar al robot.

2.2 Testeo de sensores

Es muy recomendable testear los sensores antes de programar el robot Butiá. Un método que resulta útil es:

1. Conectar el sensor a la placa USB4Butiá del robot.
2. Verificar en TurtleBots, en la paleta Butiá, que el bloque correspondiente del sensor se pone en verde y aparece su número de puerto.
3. Tomar el bloque de **por siempre** (en la paleta de operadores de flujo) y el bloque de **imprimir**, y armar un pequeño programa para imprimir por siempre los valores que retorna el sensor en cada momento. Podemos ver a continuación un ejemplo con el sensor de distancia conectado en el puerto 1:

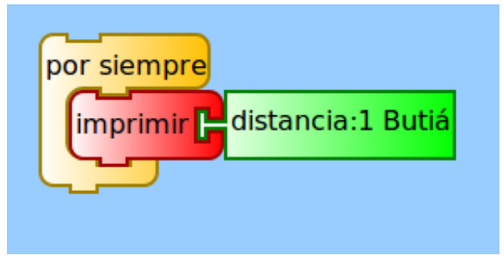


Figura 2.2.A: Ejemplo de testeo de sensor

Al ejecutar este programa, se mostrará constantemente el valor que está retornando el sensor que estamos testeando. Es importante aclarar que (excepto el sensor de contacto) los sensores del kit básico devuelven valores que varían constantemente, por más que puedan estar sensando en un entorno que no cambia. Por ejemplo si se mantiene al sensor de distancia frente a un objeto que no se está moviendo (que no varía la distancia al sensor), dicho sensor devuelve un valor determinado que varía levemente, por ejemplo retorna 33400, 33200, 32900, 34000…

A continuación se enumeran algunos posibles errores que pueden surgir al testear un sensor:

- Retorna -1.
- Retorna un valor que siempre es igual y no varía (Excepto el sensor botón que sólo devuelve 0 o 1).
- Retorna valores que varían demasiado entre sí. Por ejemplo 1200 y luego 49500.
- No funciona como corresponde (en la próxima sección se muestra cómo deberían funcionar los sensores).
- El bloque del sensor no se pone en verde aunque esté todo correctamente conectado.

Los errores mencionados se pueden deber a que el sensor está roto, o que el cable con el cual se conectó esté roto, o a una falla de TurtleBots. Algunas formas de atacar estos problemas son:

- Testear otro sensor para corroborar que lo que falla es el sensor actual.
- Cambiar de cable de sensores.
- Reiniciar TurtleBots.
- Cambiar de puerto de sensor.
- Reiniciar la computadora con el robot conectado a la misma.

2.3 Fichas de encastre y uniones

Una muy buena cualidad en el diseño del robot Butiá, es que los sensores podemos colocarlos en el robot en el punto que se desee, y orientados hacia donde se desee. Elegir qué sensores usar, y dónde colocarlos va a depender del problema que intentemos resolver, o de la prueba que se desee realizar. Algo que nos puede ayudar mucho en esta parte es el uso de las fichas de encastre y las fichas de uniones, que se muestran en la siguiente imagen:

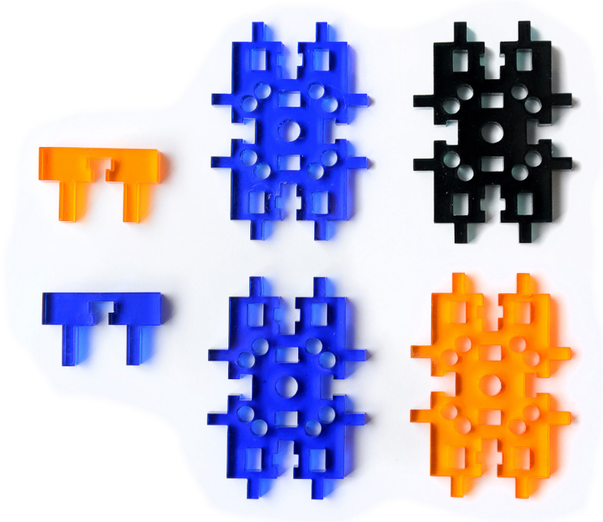


Figura 2.3.A: Fichas de encastre y fichas de uniones

Como su nombre lo indica podemos encastrar las fichas y unirlas, usando tornillos y tuercas de plástico de manera de formar diversas estructuras que nos ayudan a colocar los sensores como nos quede mejor. Por ejemplo si queremos reconocer en el piso el color blanco, podemos colocar con las fichas de encastre un sensor de escala de grises apuntando hacia abajo, o si queremos colocar un sensor botón apuntando hacia arriba podremos hacerlo también. Las fichas de encastre y las uniones son entonces piezas que facilitan el uso del robot Butiá y brindan una inmensa gama de combinaciones para colocarle sus sensores.

¿Qué valores devuelven los bloques de sensores? Podemos observar que tienen la misma forma que los bloques de números de la paleta de operadores numéricos, es decir la "T acostada" en su lado izquierdo. Por lo tanto devuelven valores numéricos. En esta sección se ve entonces los rangos de los números que devuelven y cómo podemos interpretar dichos valores para transformarlos en información útil para nuestros programas.

2.4 Sensor de contacto

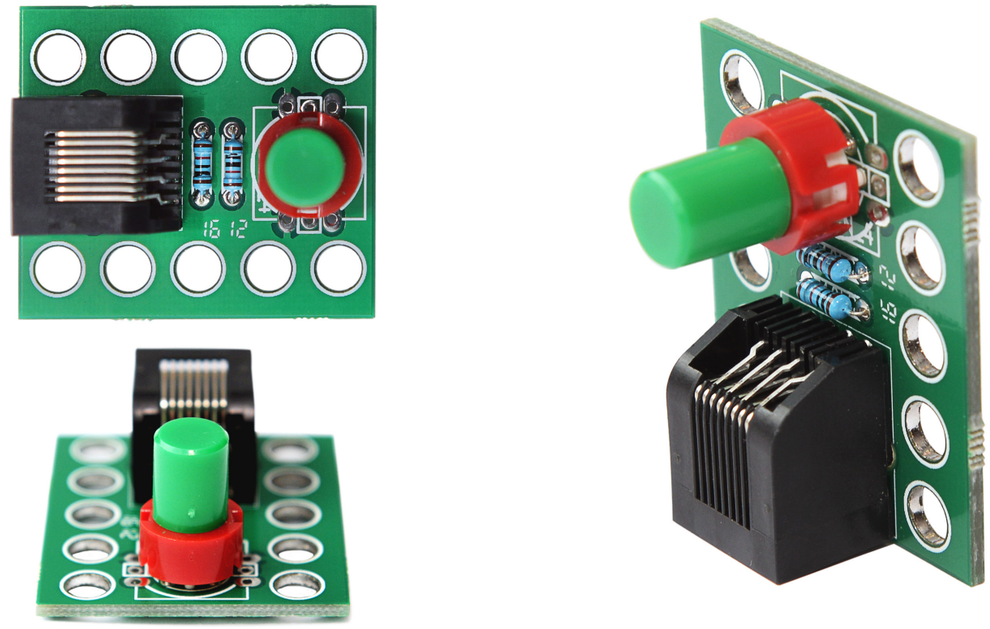


Figura 2.4.A: Sensor de contacto

El sensor de contacto, también conocido como sensor botón es simplemente un interruptor que le permite saber al robot si lo están presionando o no. Es un sensor digital, que devuelve dos valores posibles:

- 1 si está siendo presionado.
- 0 en caso contrario.

Como se muestra en la sección de testeo, una forma de probarlo es ejecutar el programa y presionar y dejar de presionar el botón para ver si retorna 1 y 0 respectivamente.

2.5 Sensor de luz

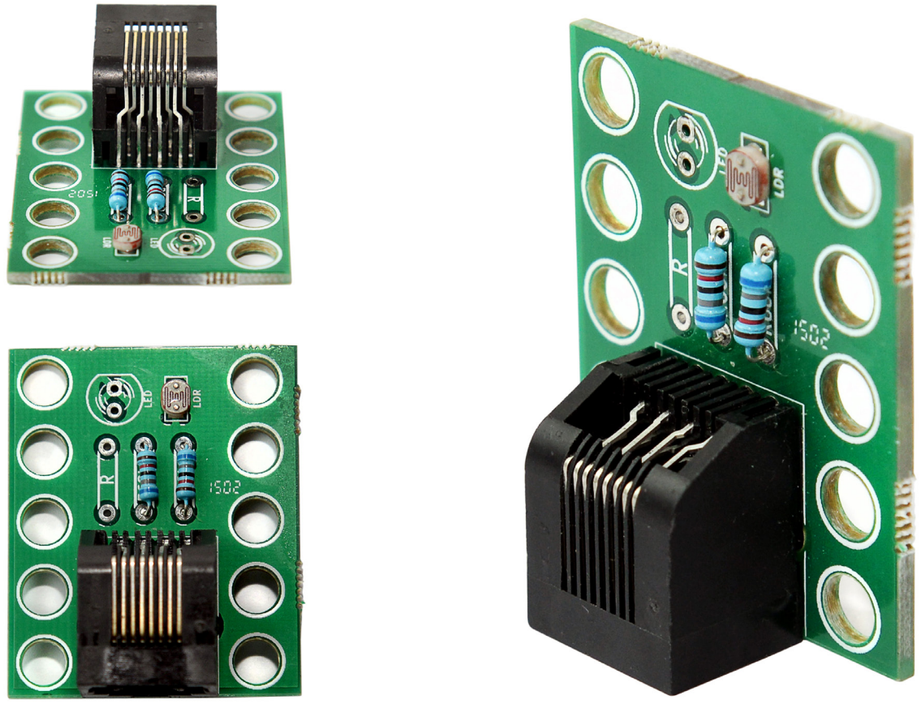


Figura 2.5.A: Sensor de luz

El sensor de luz mide la intensidad de la luz, es decir que nos da una noción de cuánta luz hay en el entorno del robot. Retorna un valor entre 0 y 65535. Mientras más luz recibe el sensor más alto es el valor que retorna, es decir 0 es oscuridad y luego aumenta dicho valor a medida de que sube la intensidad de la luz a la que está expuesto. Por ejemplo, si iluminamos el sensor con una linterna de cerca, retorna valores en el entorno de los 62500, y si tomamos el sensor y lo tapamos con la mano, vemos como baja al entorno de los 7600. Los valores mencionados son a modo de ejemplo, por supuesto que lo que retorne depende de muchos factores más, por ejemplo la luz del ambiente, y la intensidad de la linterna, etc.
Como se muestra en la sección de testeo, una forma de probarlo es ejecutar el programa e iluminar el sensor (por ejemplo con una linterna) y corroborar que devuelve valores elevados, luego taparlo y corroborar que los valores de retorno son bajos.

2.6 Sensor de distancia

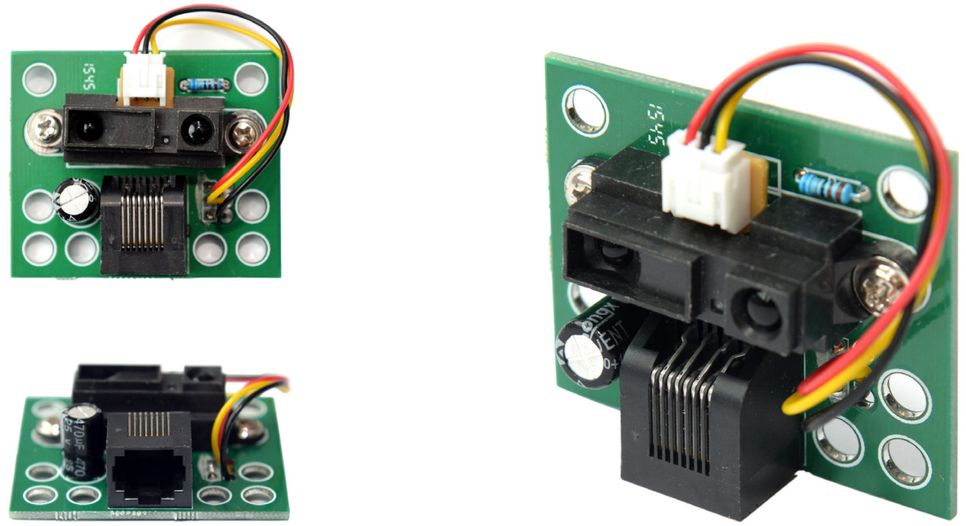


Figura 2.6.A: Sensor de distancia

El sensor de distancia retorna una noción de distancia del objeto al que apunta. Lo ideal sería que nos devuelva exactamente la distancia al objeto en metros o centímetros, pero esto es imposible en la realidad, ya que el robot Butiá es un robot de hardware libre y no se encuentra dentro de sus objetivos realizar mediciones exactas, sino que sea un robot educativo accesible que permita al usuario ser también desarrollador. Los valores que retorna son entre 0 y 65535 al igual que el sensor de luz, pero esta vez nos da una idea sobre la distancia hacia un objeto. Es decir que si medimos el sensor a una distancia determinada de un objeto, nos devolverá valores que crecerán a medida que alejemos al sensor de dicho objeto (y se reducirán a medida que lo acerquemos).
Se puede tener la idea de que si colocamos un objeto pegado al sensor de distancia nos debería devolver 0; esto no es cierto por cómo está construido. Existe un punto cercano al sensor en el cual a partir de dicho punto o más cerca, nos retorna valores aleatorios porque ya no está midiendo correctamente. Podemos verlo como un punto ciego del sensor. Por otro lado, lejos del sensor existe un punto en el cual a partir de dicho punto o más lejos, el sensor devuelve valores altos (cercanos a 65535) porque el objeto próximo está demasiado lejos.
En conclusión, el sensor de distancia funciona dentro de una intervalo que no comienza a una distancia 0 del sensor. Es decir mide la distancia de los objetos que no se encuentran ni demasiado cercanos, ni demasiado alejados.
Como se muestra en la sección de testeo, una forma de probarlo es ejecutar el programa y alejar y acercar un objeto (preferentemente de superficie plana) para corroborar que los valores que retorna crecen y decrecen respectivamente.

2.7 Sensor de escala de grises

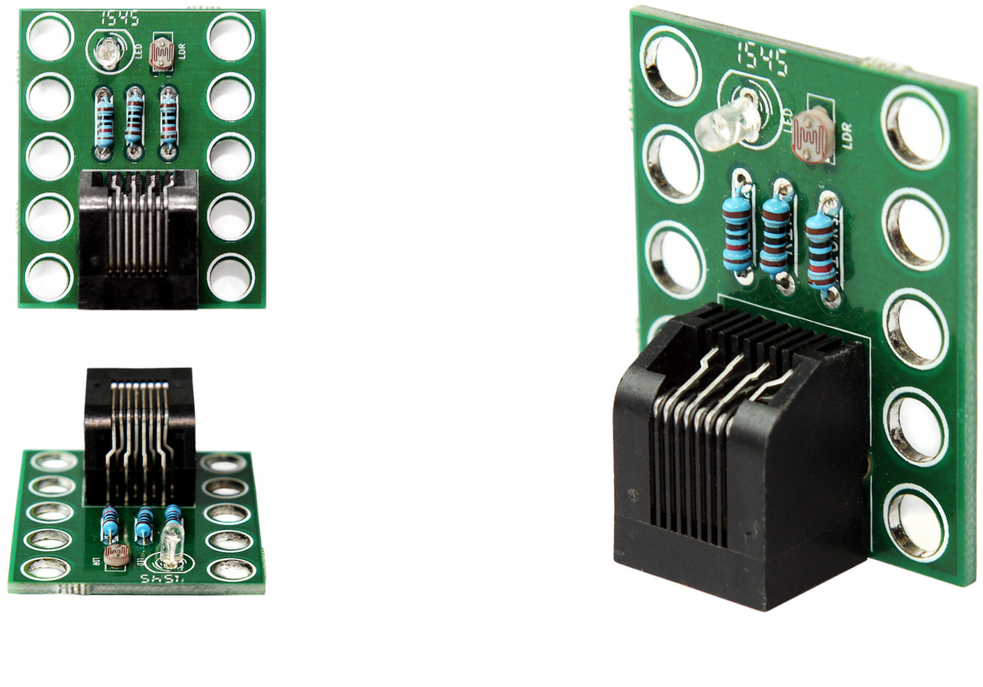


Figura 2.7.A: Sensor de escala de grises

El sensor de escala de grises sirve para obtener información de cuán blanca o cuán negra es la superficie del objeto que se está sensando. Como su nombre lo indica, devuelve valores correspondientes a la escala de grises, es decir que si sensamos una superficie blanca retorna valores pequeños, que van a crecer a medida que vaya sensando superficies más oscuras. El intervalo es también de 0 a 65535, y mientras más blanco sea lo que se está sensando, más pequeño serán los valores.

Como se muestra en la sección de testeo, una forma de probarlo es ejecutar el programa y acercar el sensor a una superficie blanca y corroborar que devuelva valores pequeños, y luego a una superficie negra y corroborar que devuelva valores altos. Es importante que exista al menos una diferencia aproximada a 10000 entre los valores que devuelve en una superficie blanca, y los que devuelve en una superficie negra, la razón de esto se explica más adelante en el ejemplo del seguidor de líneas.

3 Ejemplos de uso del robot Butiá

En la presente sección se muestran ejemplos de uso del robot Butiá, los cuales pueden posteriormente variarse para formar nuevos desafíos. Se presentan pequeños problemas que se pueden resolver utilizando al robot, y una posible solución tanto desde el punto de vista del programa en TurtleBots como de la disposición de los sensores. Los desafíos que se muestran pueden resolverse con los conocimientos básicos que se presentan en este manual.

3.1 Cuadrado

3.1.1 Desafío

El robot Butiá debe moverse describiendo la trayectoria de un cuadrado. Es decir que si lo miramos desde arriba el robot debe comenzar a moverse describiendo una trayectoria con la forma de un cuadrado y luego detenerse.

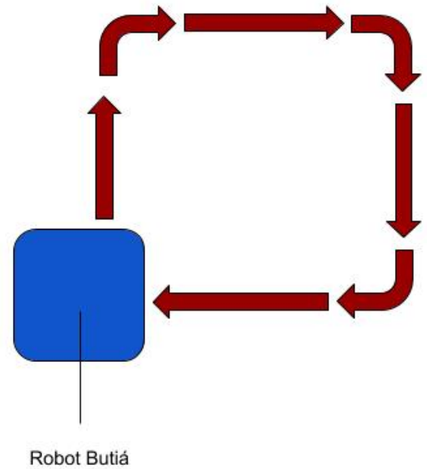


Figura 3.1.1.A: Desafío de cuadrado

3.1.2 Solución

Para resolver este problema, el robot no necesita sensores, ya que no necesita información de su entorno. Sólo debe moverse describiendo la trayectoria deseada. Como vimos en la paleta Butiá, podemos mandar las órdenes a los motores de moverse hacia adelante, atrás, girar a la izquierda

y girar a la derecha. Por lo tanto para describir un cuadrado nos alcanza con enviar órdenes de ir hacia adelante y hacia la izquierda (o a la derecha, dependendiendo hacia donde queremos que gire) de manera de que describa los lados del cuadrado. Un pseudocódigo para resolver el problema sería:

```
Adelante
Girar 90 grados hacia la izquierda
Adelante
Girar 90 grados hacia la izquierda
Adelante
Girar 90 grados hacia la izquierda
Adelante
Detener
```

Este pseudocódigo nos plantea las siguientes preguntas:

- ¿ Cuánto se desplaza hacia adelante el robot Butiá?
- ¿Cómo hacer que gire exactamente 90 grados?

Dado que no podemos indicarle al robot cuántos metros avanzar ni cuántos grados girar, ambas preguntas se responden con el uso del mismo bloque: el bloque **esperar**. Para avanzar se indica cuántos segundos se quiere avanzar, (que es lo mismo que decir cuántos segundos demora el robot en moverse describiendo un lado del cuadrado). Para girar 90 grados debe testearse el robot, es decir, lo ponemos a girar y medimos cuánto tiempo demora en girar 90 grados. Éstas medidas nunca serán exactas y tampoco es la intención. Además el robot nunca se moverá describiendo un cuadrado exacto, por miles de factores que inciden: la batería del robot, la potencia de los motores, el piso sobre el que se mueve, la fricción de las ruedas, etc. Sin embargo, al medir estos tiempos, podemos completar el pseudocódigo con el bloque esperar y resolver el desafío. Como ejemplo vamos a suponer que se quiere que avance 3 segundos por lado, y que luego de medir demora aproximadamente 0.7 segundos en girar 90 grados sobre sí mismo, por lo que el pseudocódigo quedaría de la siguiente forma:

```
Adelante durante 3 segundos
Girar a la izquierda durante 0.7 segundos
Adelante durante 3 segundos
Girar a la izquierda durante 0.7 segundos
Adelante durante 3 segundos
Girar a la izquierda durante 0.7 segundos
Adelante durante 3 segundos
Detener
```


A continuación se muestra el código del programa en TurtleBots:

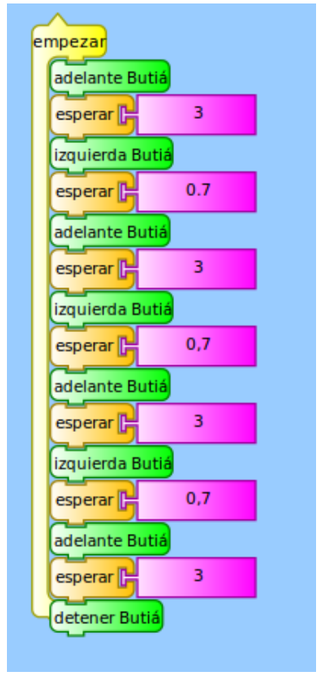


Figura 3.1.2.A: Ejemplo de solución de desafío del cuadrado

Con este código deberíamos poder colocar el robot en el piso, ejecutar el programa y resolver el desafío. Si observamos bien, hay porciones de código que se repiten (la de avanzar y la de girar). Por lo tanto podemos hacer uso del bloque **repetir** de manera de resolver el mismo problema, pero escribiendo el programa más prolijo y explotando las cualidades de las estructuras de control:

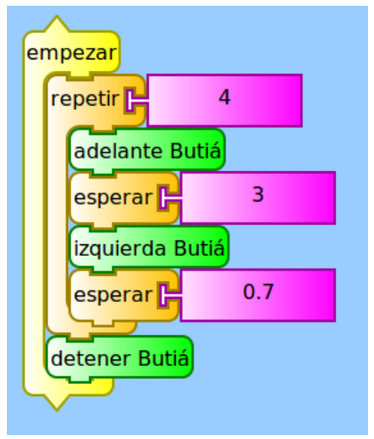


Figura 3.1.2.B: Ejemplo de solución de desafío del cuadrado con repetir

3.2 Seguidor de líneas

3.2.1 Desafío

El robot Butiá debe seguir un camino negro, sobre un fondo blanco. A continuación se muestran imágenes de ejemplo de las pistas sobre las cuales se plantea el desafío.

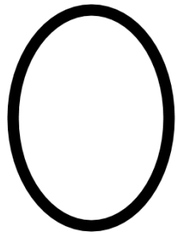


Figura 3.2.1.A: Pista ovalada

Figura 3.2.1.B: Pista genérica

No es casualidad que el camino sea de color negro y el resto de color blanco. Se plantea el desafío de esta forma para poder hacer uso del sensor de escala de grises. El robot Butiá deberá usar dicho sensor para poder distinguir si va bien por el camino, o si se está saliendo de la trayectoria deseada.

3.2.2 Calibrando el sensor de escala de grises

Como vimos en la sección anterior, el sensor de escala de grises nos devuelve un valor entre 0 y 65535, y para resolver este desafío queremos que el robot use dicho sensor para distinguir cuándo está sensando color negro y cuándo está sensando color blanco. Al testear sensores podemos notar que los valores que devuelven nunca son constantes y siempre varían aunque sea de forma leve (excepto para el sensor boton). Una buena manera de lograr lo deseado es encontrar un umbral, es decir un valor X dentro del intervalo con el cual podemos decir: si el sensor retorna un valor mayor que X, entonces está sensando el color negro, de lo contrario está sensando el color blanco. ¿Cómo encontramos este valor? Se explica a continuación paso a paso:

1. Tomamos el sensor que queremos calibrar y armamos el programa de testeo (como se indica en la sección de testeo de sensores).
2. Colocamos el sensor sensando blanco y ejecutamos el programa, luego tomamos una pequeña muestra de los valores que retorna el sensor (por ejemplo anotamos 5 valores retornados).
3. Calculamos el promedio de esa muestra, llamémosle B a ese promedio.
4. Análogamente al punto 2, colocamos el sensor sensando negro y ejecutamos el programa, luego tomamos una pequeña muestra de los valores que retorna el sensor.
5. Calculamos el promedio de esa muestra, llamémosle N a ese promedio.
6. Lo ideal es que el umbral X se encuentre a la mitad de la distancia entre B y N. Por eso se dijo anteriormente que es deseable que exista una diferencia aproximada a 10000 entre los valores de blanco y los de negro. Entonces X se puede calcular de la siguiente forma:

$$X = (B + N) / 2$$

Nota: Cada sensor tiene su umbral, y dicho umbral puede variar según el material del piso, la luz del entorno, el reflejo en el piso, etc. Por lo tanto el proceso de calibrar hay que hacerlo para cada sensor de escala de grises que usemos, y cada vez que lo vayamos a usar, dado que no es un valor que se halla una vez y se mantiene.

Veamos un ejemplo de calibración del sensor de escala de grises. Supongamos que tenemos un sensor de escala de grises conectado al robot:

1. Colocamos el sensor sobre el color blanco y supongamos que con el programa de testeo retorna los valores: 11400, 13000, 12600, 11500, 11000.
2. Calculamos el promedio de los valores en color blanco:

$$B = (11400 + 13000 + 12600 + 11500 + 11000)/5 = 11900$$

3. Ahora repetimos el proceso sobre el color negro, y supongamos que retorna los valores: 22100, 24000, 21600, 23400, 22500.
4. Calculamos el promedio de los valores en color negro:

$$N = (22100 + 24000 + 21600 + 23400 + 22500)/5 = 22720$$

5. Ahora con estos dos valores podemos hallar el umbral X:

$$X = (B + N) / 2 = (11900 + 22720)/2 = 17310$$

Ahora, dentro de nuestro programa, para hacer la pregunta: ¿El sensor está sensando negro? Que es equivalente a preguntar: ¿El sensor está devolviendo un valor mayor al umbral? Lo codificamos de la siguiente manera(supongamos que el sensor está en el puerto 2):

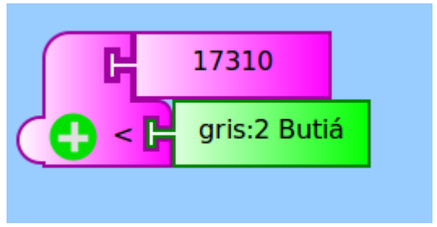


Figura 3.2.2.A: Ejemplo de consulta con umbral

3.2.3 Pista ovalada

A continuación se muestra una solución para este tipo de pista. En la parte delantera del robot colocamos un sensor apuntando hacia abajo (porque el blanco y negro se van a sensar en el piso) usando las piezas de encastre, como se muestra en la foto a continuación.

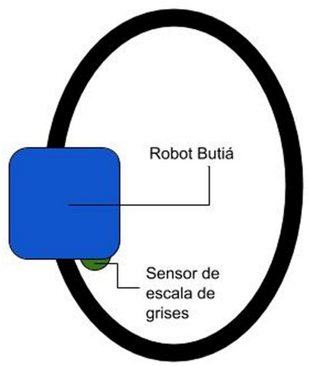


Figura 3.2.3.A: Butiá en pista ovalada

Mirando desde arriba, podemos ver que el sensor de escala de grises queda en el lado interior de la pista ovalada, sensando el color blanco. Por lo tanto el desafío puede resolverse, simplemente haciendo que el programa esté constantemente consultando sobre el color que sensa, y tomando las siguientes decisiones:

```
Si se ve color blanco:
        Avanzar
Sino(es decir, se ve color negro):
        Girar hacia la derecha
```

Suponiendo que calibramos el sensor y obtenemos un umbral de **19000**, y además queremos que siempre se haga la pregunta (es decir que el robot siempre sense), el pseudocódigo queda:

```
Por siempre:
        Si sensor_gris < 19000:
                Avanzar
        Sino
                Girar hacia la derecha
```

El **por siempre** se debe a que el programa siempre debe estar sensando y consultando el valor para tomar la decisión de girar o avanzar en cada momento. Ahora veamos este código en TurtleBots (supongamos que el sensor de escala de grises está conectado en el puerto 5):

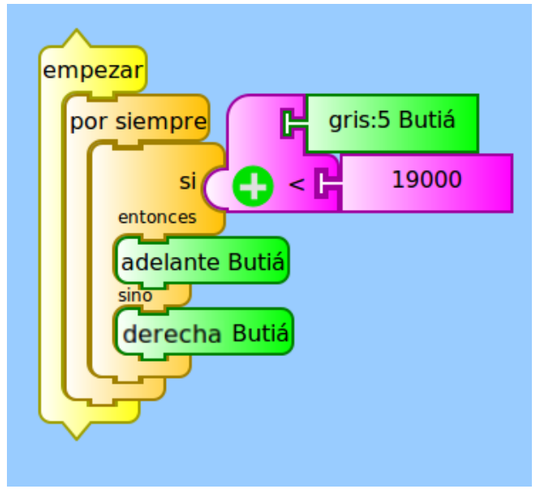


Figura 3.2.3.B: Ejemplo de solución de pista ovalada

3.2.4 Pista con forma genérica

En la pista ovalada, tenemos la ventaja de que el robot Butiá siempre debe girar para el mismo lado dada la naturaleza de la pista y de cómo colocamos el sensor. Cuando se desea que el robot Butiá siga por una pista que puede tener cualquier forma, es posible hacer una solución general, de manera de que el programa no dependa de la forma del camino a seguir. La solución que proponemos no es la única, y se propone usar dos sensores de grises. Armamos el robot como se muestra en la foto a continuación:

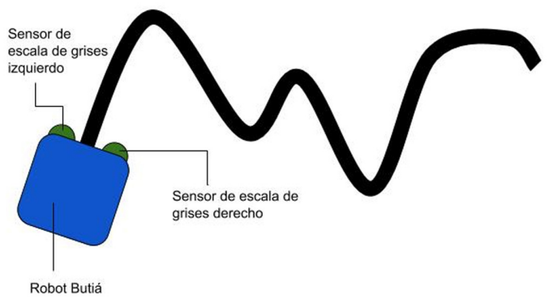


Figura 3.2.4.A: Butiá en pista genérica

Podemos ver que un sensor queda del lado izquierdo del camino, y otro del lado derecho. En el programa a construir debemos tener en cuenta ahora lo que retornan ambos sensores. En cada momento, la pregunta es: ¿Ambos sensores ven el color blanco? si esto es cierto se debe avanzar porque significa que el camino está en el medio de los dos sensores. Cuando no es cierto que ambos sensores ven el blanco, debemos distinguir cuál de ellos está sensando negro: si el sensor izquierdo está sensando negro, significa que el camino está doblando hacia la izquierda, por lo tanto hay que girar en ese sentido. Análogamente hay que girar hacia la derecha cuando es el sensor derecho el que está sensando el color negro. Notar que se asume que el camino no tiene bifurcaciones, ni forma de T, y también se asume que nunca va a pasar que ambos sensores sensen negro simultáneamente. Veamos un pseudocódigo:

```
Por siempre:
        Si ambos sensores ven blanco:
                Avanzar
        Sino:
                Si el sensor derecho ve negro:
                        Girar hacia la derecha
                Sino:
                        Girar hacia la izquierda
```

Notar de nuevo el **por siempre** para estar constantemente consultando el valor retornado. Supongamos que:

- El sensor colocado a la **izquierda** está en el **puerto 5**, y al hallar su umbral obtuvimos el valor **17500**.
- El sensor colocado a la **derecha** está en el puerto **2**, y al hallar su umbral obtuvimos el valor **22000**.

El pseudocódigo queda de la siguiente manera:

```
Por siempre:
        Si (sensor_gris_en(5) < 17500) y (sensor_gris_en(2) < 22000):
                Avanzar
        Sino:
                Si (22000 < sensor_gris_en(2)):
                        Girar hacia la derecha
                Sino:
                        Girar hacia la izquierda
```

El pseudocódigo anterior, se muestra a continuación programado en TurtleBots:

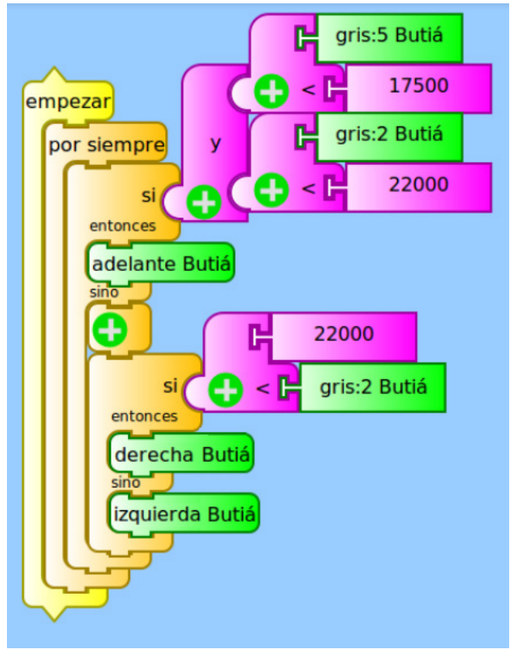


Figura 3.2.4.B: Ejemplo de solución de pista genérica

Es importante aclarar que tanto la solución de la pista ovalada, como la de la pista genérica, no son únicas. Existen otras disposiciones de los sensores, combinadas con otros programas, que pueden resolver el mismo desafío. Quedan para el lector las siguientes preguntas:

¿La solución de la pista ovalada sirve para resolver el desafío de la pista genérica?
¿La solución de la pista genérica sirve para resolver el desafío de la pista ovalada?
¿Se puede resolver el desafío de la pista genérica usando sólo un sensor?
Si en la pista ovalada doy vuelta el robot, ¿Hay que cambiar el programa?

3.3 Esquivador de obstáculos

3.3.1 Desafío

El robot Butiá debe avanzar, pero si en frente aparece un obstáculo, debe esquivarlo y seguir avanzando en otra dirección.

3.3.2 Solución

Para la solución propuesta se elige utilizar el sensor de distancia, que nos ayudará a reconocer un objeto que se aproxime frente al robot. Esto se debe a que podemos definir una distancia hacia cualquier objeto con la cual el robot debe tomar la decisión de esquivar. El sensor lo colocamos en la parte delantera del robot apuntando hacia adelante, como podemos ver en la foto:

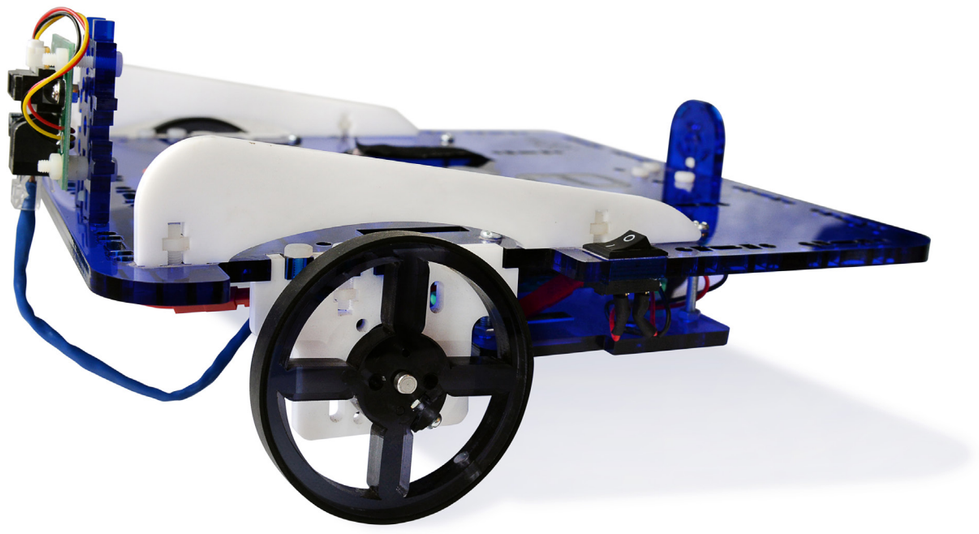


Figura 3.3.2.A: Robot con sensor distancia apuntando hacia adelante

De esta forma el robot podrá reconocer los objetos que pasen por adelante del sensor. Lo que hay que hacer entonces es calibrar para obtener el valor de la distancia deseada.

3.3.3 Calibrando el sensor de distancia

Supongamos que definimos la distancia requerida para la solución como 27 cm, lo que debemos hacer es calibrar el sensor para obtener el valor correspondiente a esa distancia. Como ya sabemos a esta altura nunca va a ser un valor exacto, pero podemos definir un umbral X con el cual podemos decir: si el sensor de distancia retorna un valor menor a X para una distancia dada, entonces estamos a esa distancia o menos del objeto que el sensor tiene por delante. Para ello hacemos algo similar que al calibrar el sensor de escala de grises:

1. Tomamos el programa de la sección de testeo y colocamos el objeto a la distancia deseada, frente al sensor de distancia.
2. Ejecutamos el programa y tomamos una muestra de las medidas que retorna (por ejemplo tomamos 5 valores)
3. El umbral X correspondiente a la distancia deseada, es el promedio de dicha muestra.

Ahora bien, ya tenemos nuestro umbral X, lo que tenemos que hacer es razonar sobre cómo debe funcionar nuestro programa: una vez que el robot detecte que se aproxima a un obstáculo, le decimos primero que se detenga y luego que retroceda y gire, para poder avanzar en otra dirección. Un pseudocódigo de la posible solución sería:

```
Por siempre:
        Si el sensor detecta una distancia menor al umbral:
                Detenerse
                Retroceder
                Girar hacia la izquierda
        Sino:
                Avanzar
```

Notar que ponemos el **por siempre** ya que esa es una pregunta que el programa debe estar siempre realizando. Como bien sabemos a esta altura, necesitamos del bloque esperar para que tengan sentido las órdenes a los motores del robot. Suponiendo que conectamos el sensor de distancia en el puerto 1, y el umbral es 34000, nos queda el siguiente pseudocódigo:

```
Por siempre:
    Si sensor_distancia_en(1) < 34000:
        Detener Butiá durante 0.5 segundos
        Retroceder durante 3 segundos
        Girar hacia la izquierda durante 2 segundos
    Sino:
        Avanzar
```

El tiempo de espera al detenerse, retroceder y girar es también a modo de ejemplo. A continuación se muestra el programa de la solución propuesta, programado en TurtleBots:

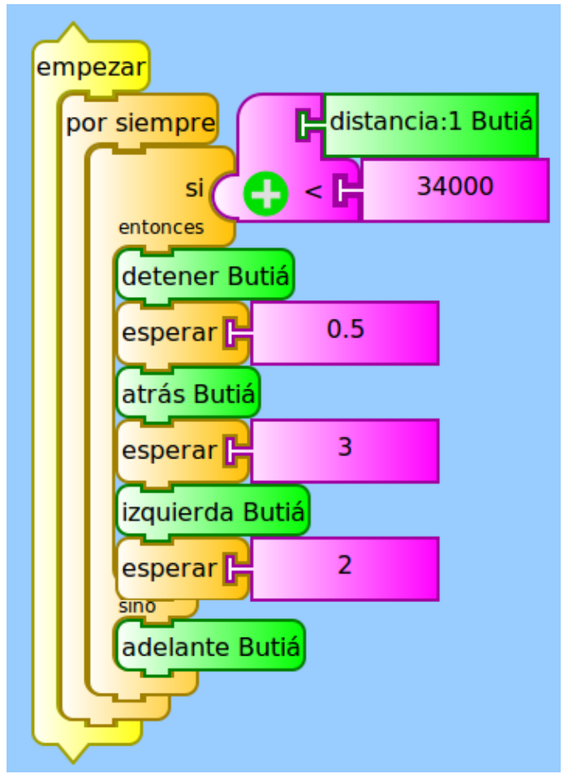


Figura 3.3.3.A: Ejemplo de solución de esquivar obstáculos

Glosario

- **Plugin** Plugin, que también puede mencionarse como plug-in, es una noción que no forma parte del diccionario de la Real Academia Española (RAE). Se trata de un concepto de la lengua inglesa que puede entenderse como "inserción" y que se emplea en el campo de la informática. Un plugin es aquella aplicación que en un programa informático, añade una funcionalidad adicional o una nueva característica al software. En español puede nombrarse al plugin como un complemento.

- **Browser** Es un navegador de Internet, un software que permite (entre otras cosas) la visualización de los contenidos que presenta una página web.

- **Plug and play** La expresión Plug-and-Play o PnP (en español "enchufar y usar") es la tecnología o cualquier avance que permite a un dispositivo informático ser conectado a una computadora sin tener que configurar, ni proporcionar parámetros a sus controladores. Esto significa que un dispositivo que posee la propiedad de PnP, al conectarlo a la computadora ya está listo para funcionar sin necesidad de configuración previa.

- **Hotplug** Hotplug es una noción que no forma parte del diccionario de la Real Academia Española (RAE). Es la capacidad de un dispositivo para ser conectado o desconectado cuando la computadora está encendida. Esto significa que no es necesario reiniciar la computadora ni la aplicación que usa al dispositivo, para que este último pueda desconectarse y conectarse, y funcione correctamente.

- **Actuador** Un actuador es un dispositivo que utiliza un robot para modificar su entorno. Puede utilizarlo para moverse, emitir un sonido, desplazar un objeto, o cualquier acción que genere un cambio en el entorno.

- **Pseudocódigo** Un pseudocódigo es la descripción de un programa o algoritmo a un nivel entre el lenguaje natural, y un lenguaje formal de computación. Se emplea cuando se pretende describir un algoritmo sin la necesidad de difundir cuáles son sus principios básicos. De esta manera, un ser humano encontrará mayores facilidades para comprender el mensaje, a diferencia de lo que ocurriría si estuviese frente a un lenguaje de programación real.

- **Sensar** Sensar es la acción que ejecuta un sensor, es decir, tomar medidas del entorno utilizando un sensor.


CEN
TRO
MEC