

Apostila Robô Fun

Como Utilizar a Interface

Projeto CNPq-Vale – Grupo PETECO/DAINF/UTFPR-CT

SUMÁRIO

| | |
|--|-----------|
| INTRODUÇÃO | 3 |
| INTERFACE DE PROGRAMAÇÃO DO ROBÔ F | 4 |
| DEFINIÇÕES BÁSICAS | 4 |
| COMANDOS DA API DO ROBÔ F | 4 |
| TUTORIAL DE COMO UTILIZAR A INTERFACE | 10 |
| 1 - BOTÕES DE ORGANIZAÇÃO | 11 |
| 2 – ABA DE FEEDBACK | 12 |
| 3 – ÁREA DE DESENVOLVIMENTO | 13 |
| 4 – BOTÕES DE DESENVOLVIMENTO | 15 |
| PENSAMENTO COMPUTACIONAL | 17 |

INTRODUÇÃO

Este material serve de apoio ao curso introdutório de robótica e computação baseado na interface de programação JIFI. Durante o curso serão abordados os assuntos básicos de programação que são ensinados no início dos cursos de “Sistemas de Informação” e “Engenharia de Computação” juntamente com alguns tópicos iniciais de eletrônica.

Basicamente a apostila reúne alguns problemas clássicos de programação moldados de forma que possam ser solucionados usando o Robô F. Para resolver os problemas o estudante pode usar o software de programação visual desenvolvido pelo PET-ECO, uma interface gráfica que permite organizar comandos na forma de um fluxograma que então é enviado ao robô, que executa os passos.

Na interface também há a opção de simular o funcionamento do Robô F em um ambiente controlado, além disso, podem-se monitorar as leituras de sensores e o posicionamento tanto do robô simulado como do real.

INTERFACE DE PROGRAMAÇÃO DO ROBÔ F

Definições Básicas

A linguagem de programação visual é composta de:

1. Comandos disponíveis na API de programação dos robôs (ver abaixo)
2. Bloco de comandos sequenciais;
3. Loop: assumir que para o módulo 1 só existe *while*
4. Função com argumentos;
5. Variáveis sem tipificação;
6. Variáveis vetoriais;
7. Operadores
 - a. Aritméticos básicos (4 operações)
 - b. Relacionais (maior, menor, maior ou igual, menor ou igual, diferente).
 - c. Lógicos (and, or).

Comandos da API do Robô F

move

wait

rotate

var

read

print

if...else

while

move

```
move ( r1, r2 );
```

Move o robô, sendo *r1* e *r2* a roda esquerda e direita respectivamente. Cada roda recebe um valor inteiro de velocidade relativa, sendo 0 a roda parada, 100 a velocidade máxima para frente e -100 a velocidade máxima para trás do motor.

Exemplo

```
move (100, 100);
```

Move o robô em linha reta para frente com velocidade máxima.



wait

```
wait ( t );
```

wait (t) serve para controlar o tempo de um comando. O tempo é regulado pelo parâmetro inteiro t, e é medido em milissegundos.

Exemplo

```
move (100, 100);  
wait (1000);  
move (0, 0);
```

Faz o robô acionar os motores em velocidade máxima para frente por 1 segundo e depois parar.

rotate

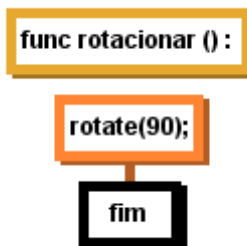
```
rotate ( a );
```

Rotaciona o robô em torno de seu próprio eixo ao fazer as rodas girarem em sentidos opostos na mesma velocidade. Recebe o parâmetro angular inteiro *a* (-360 a 360). Valores positivos rotacionam o robô no sentido horário e valores negativos rotacionam o robô no sentido anti-horário.

Exemplo

```
rotate (90);
```

Rotaciona o robô 90° no sentido horário.



var

```
var <nome>;
```

Cria uma variável , ou seja, reserva um espaço de memória para guardar uma variável de qualquer tipo (números inteiros, reais, caracteres, etc.). O comando também aceita vetores de tamanho variável.

Exemplo

```
var x = 0;
```

Cria uma variável de nome *x* e atribui o valor zero a ela.

```
var v;  
v = [10,20];
```

Cria um vetor com 2 posições preenchidas com 10 e 20.

```
func variavel () :
```

```
var x = 1;
```

```
fim
```

read

```
read ( Device, var );
```

Comando usado para guardar valor de uma variável lida pelo sensor. Recebe um parâmetro *Device*, que é um dos dispositivos do robô (bússola, sensor de distância e sensores de refletância) e guarda o valor recebido na variável *var*.

Exemplo

```
var y;  
read (Bussola, y);
```

Cria uma variável *y* e guarda o valor lido da bússola em *y*, no caso um número inteiro representando um ângulo (0 a 359°).



print

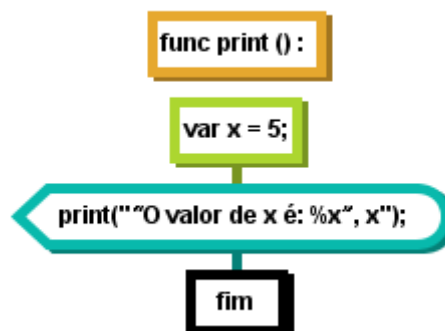
```
print ( <formato>, [var1, ..., varN] )
```

Exibe uma mensagem na tela. O formato é delimitado por aspas duplas (") e pode conter a expressão `%v`, que será substituída pela variável fornecida como parâmetro.

Exemplo

```
var x = 5;  
print ("O valor de x é: %x", x);
```

Exibe a mensagem: "O valor de x é: 5".



if...else

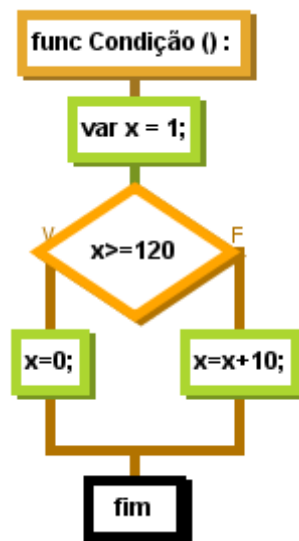
If (condição) { ... } else { ... }

O comando *if* (Se), serve para fazer um caminho alternativo . Irá executar um bloco de comandos apenas se determinada condição for satisfeita. O comando *else* (Senão) executa um bloco caso a condição não seja satisfeita

Exemplo

```
if (x >= 120)
{
    x = 0;
}
else
{
    x = x + 10;
}
```

O valor da variável *x* será zerado se a variável apresentar um valor maior ou igual a 120. Caso contrário a variável é incrementada em 10 unidades.



while

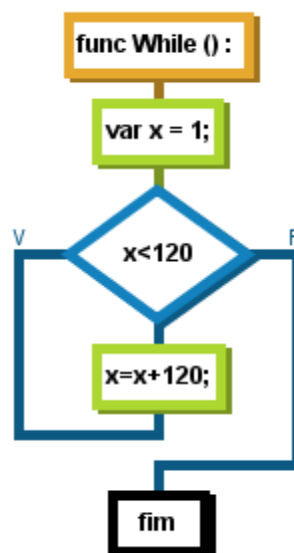
```
while ( condição )
```

Comando de loop de repetição. Irá executar um bloco de comandos enquanto uma determinada condição for satisfeita.

Exemplo

```
while (x < 120)
{
    x = x + 10;
}
```

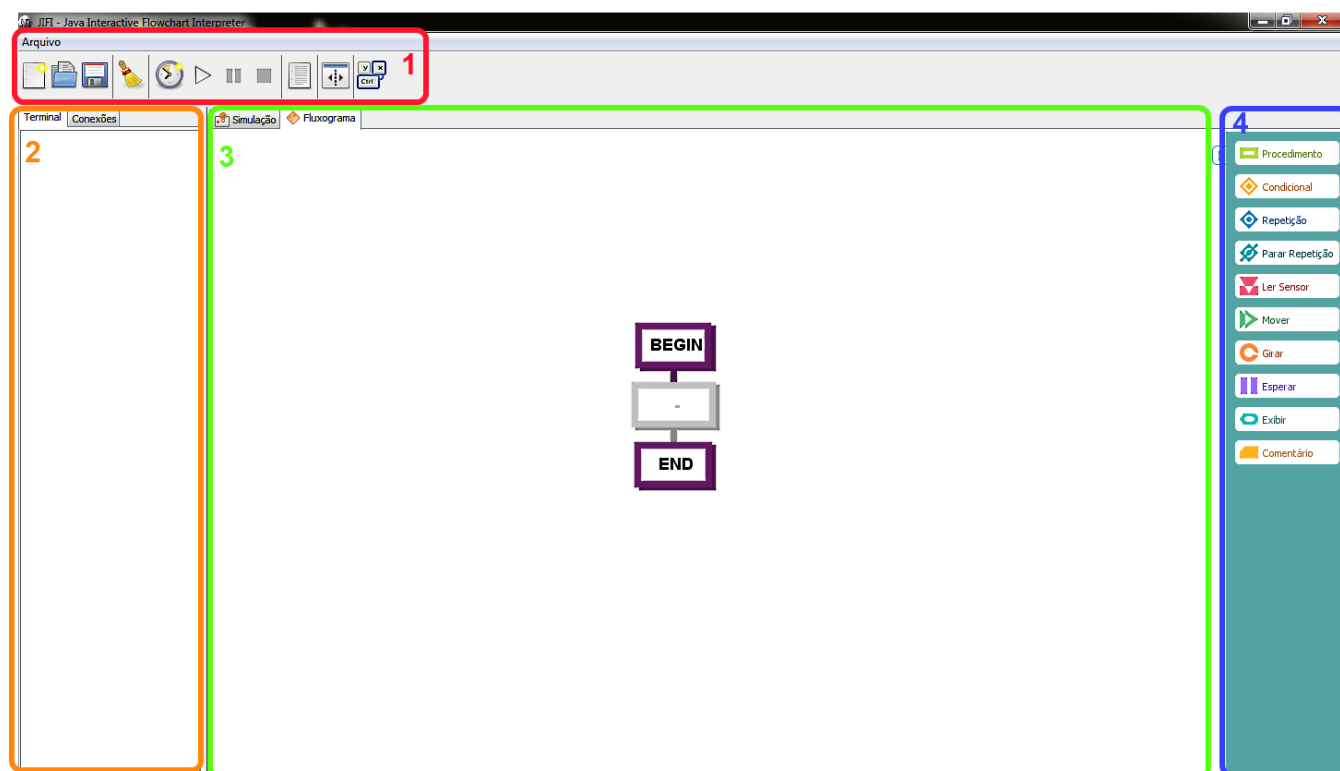
Enquanto o valor da variável x é menor que 120 a variável tem seu valor incrementado em 10 unidades.



TUTORIAL DE COMO UTILIZAR A INTERFACE

A plataforma JIFI (Java Interactive Flowchart Interpreter) é um software de programação visual desenvolvido pelo PET-ECO, ou seja, uma interface gráfica que permite organizar comandos na forma de um fluxograma que então é enviado ao robô, que então executa esses comandos.

Para um melhor entendimento de como manejar a plataforma JIFI, este tutorial foi dividido em quatro partes:



1 - Botões de Organização

2 - Aba de Feedback

3 - Área de Desenvolvimento

4 - Botões de Desenvolvimento

1 - Botões de Organização

Estes botões são de suma importância ao usuário, portanto, com o intuito de ensinar com cautela suas funcionalidades, são apresentados detalhadamente:



Novo Arquivo: Fecha o projeto atual, limpa a simulação, e cria um programa vazio.



Abrir: Abre um projeto salvo.



Salvar: Salva o projeto atual (ambiente e código).



Limpar Simulação: Remove todas as linhas colocadas na simulação (esta simulação não pode ser desfeita).



Diminuir Tempo: Coloca uma espera a cada comando executado permitindo visualizar a execução do programa lentamente.



Executar: Executa o programa criado.



Pausar: Pausa a execução do programa.



Parar: Interrompe a execução do programa e retorna ao primeiro comando.



Converter: Transforma o fluxograma em código fonte e vice-versa.



Visão Lado a Lado: Permite visualizar a simulação e o programa lado a lado, clique outra vez para separa-los.



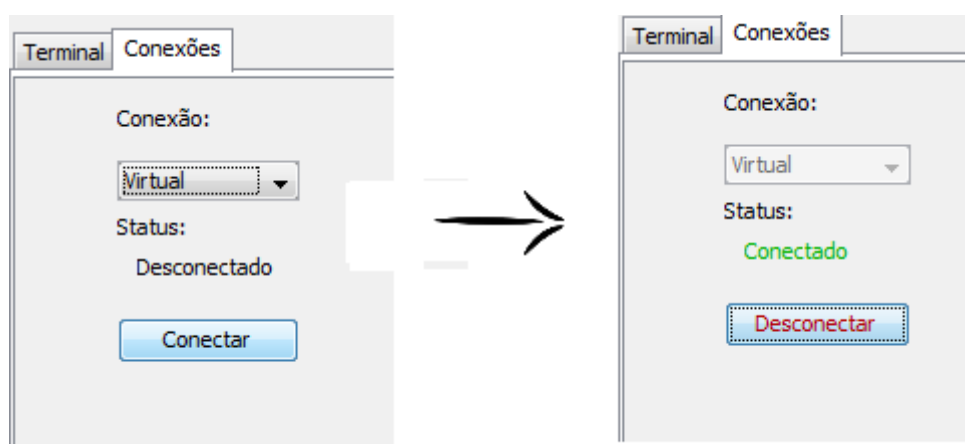
Atalhos do Programa: Exibe uma lista dos principais atalhos do programa.

2 – Aba de Feedback

Esta aba apresenta um feedback ao usuário, ou seja, serve para reportar certas informações importantes, como por exemplo se o robô está conectado, ou se o sinal de rádio está sendo enviado.

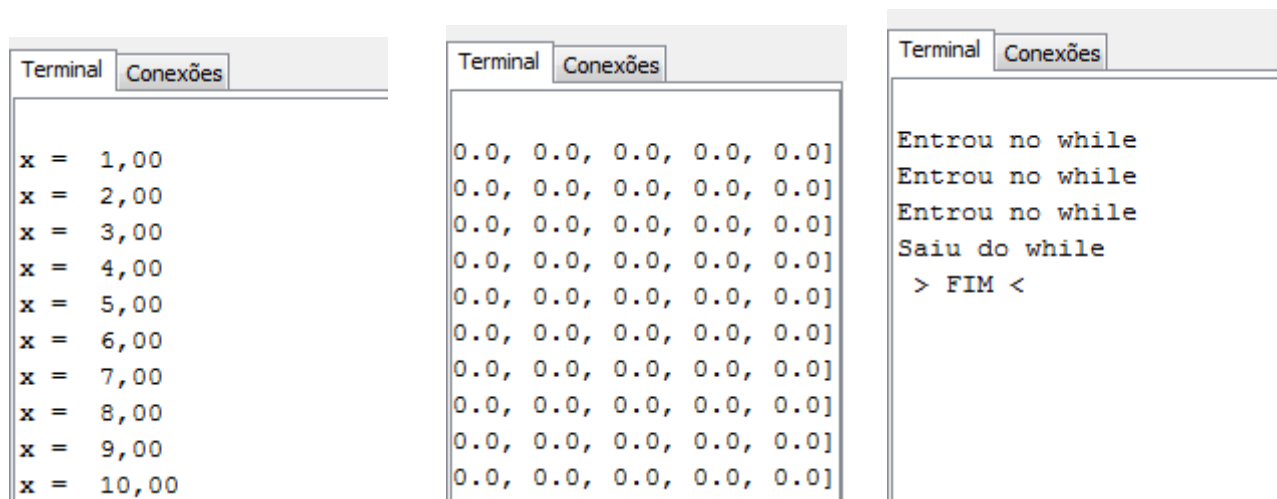
A aba permite se comunicar com o robô real e acompanhar os pacotes enviados, perdidos e recebidos.

Para conectar a simulação virtual ou o robô real basta escolher a opção de conexão e então clicar em conectar. Vale a pena ressaltar que a simulação virtual também é realizada quando o tipo de conexão escolhida é a da porta USB (robô).



(ADICIONAR PRINT SCREEN DO ROBO CONECTADO – SINAL ENVIANDO)

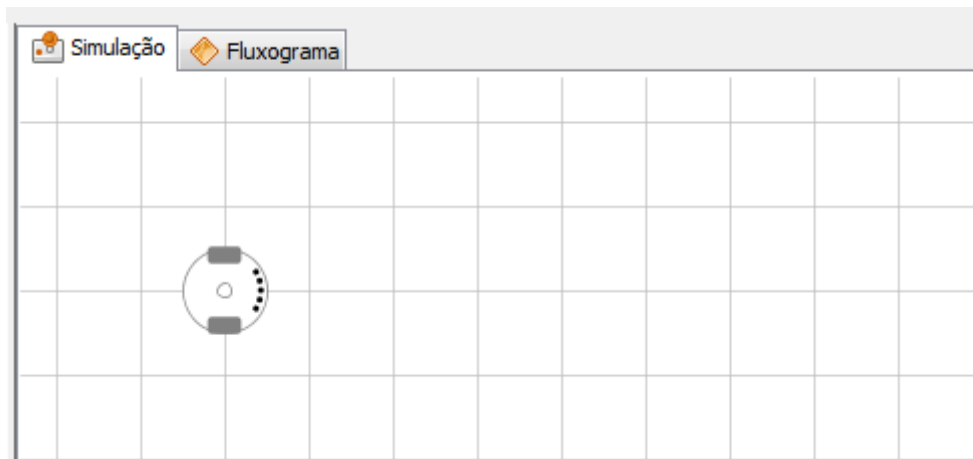
A aba Terminal serve para visualizar mensagens enviadas pelo robô (comando “Exibir”).



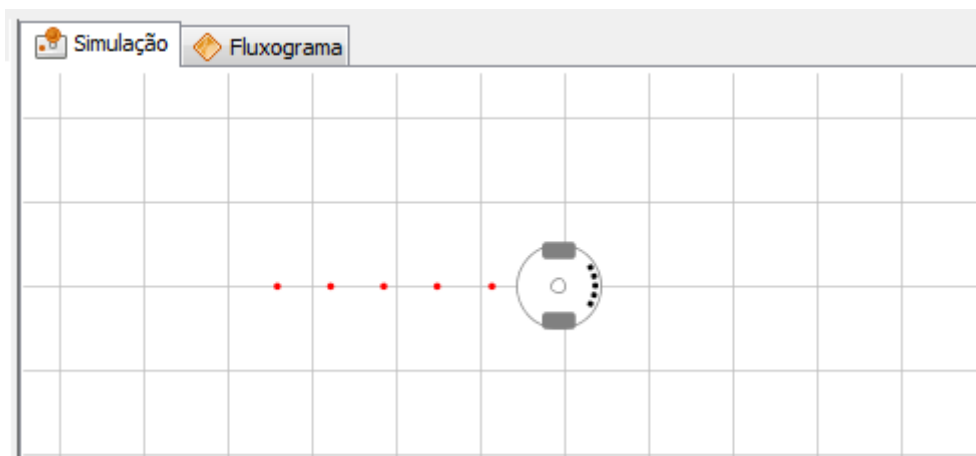
Mensagens como as apresentadas acima são muito úteis na programação, pois servem como um guia já que informam ao usuário dados referentes ao que realmente está ocorrendo no programa.

3 – Área de Desenvolvimento

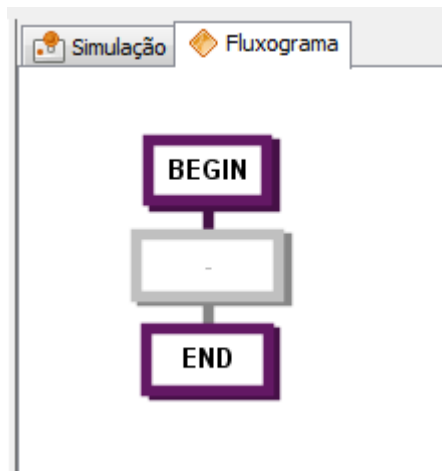
A aba de simulação é uma área que permite visualizar o robô se movendo e interagindo com obstáculos conforme programado.



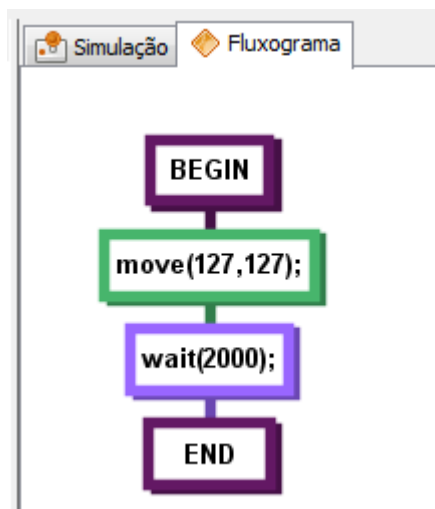
Seguindo um exemplo básico em que o robô anda em linha reta por um determinado tempo à uma certa velocidade, a execução é apresentada assim:



A aba fluxograma é o editor de código, que permite definir o comportamento do robô em sua interação com o meio ambiente. Antes de adicionar algum bloco de comando ou escrever por código fonte o fluxograma se encontra nesse estado:




No caso, BEGIN significa o começo do programa e END significa o final. Os blocos de comando então são adicionados nesse espaço vazio. A seguir o mesmo exemplo apresentado para a simulação em que o robô segue uma linha reta, porém agora sua forma em fluxograma:





4 – Botões de Desenvolvimento


Para adicionar linhas e paredes na simulação ou então blocos de comando e comentários ao fluxograma são utilizados os botões de desenvolvimento.

Na aba de simulação os botões encontrados são:

 **Linha** : Linha preta colocada no chão que pode ser detectada pelo sensor de refletância.

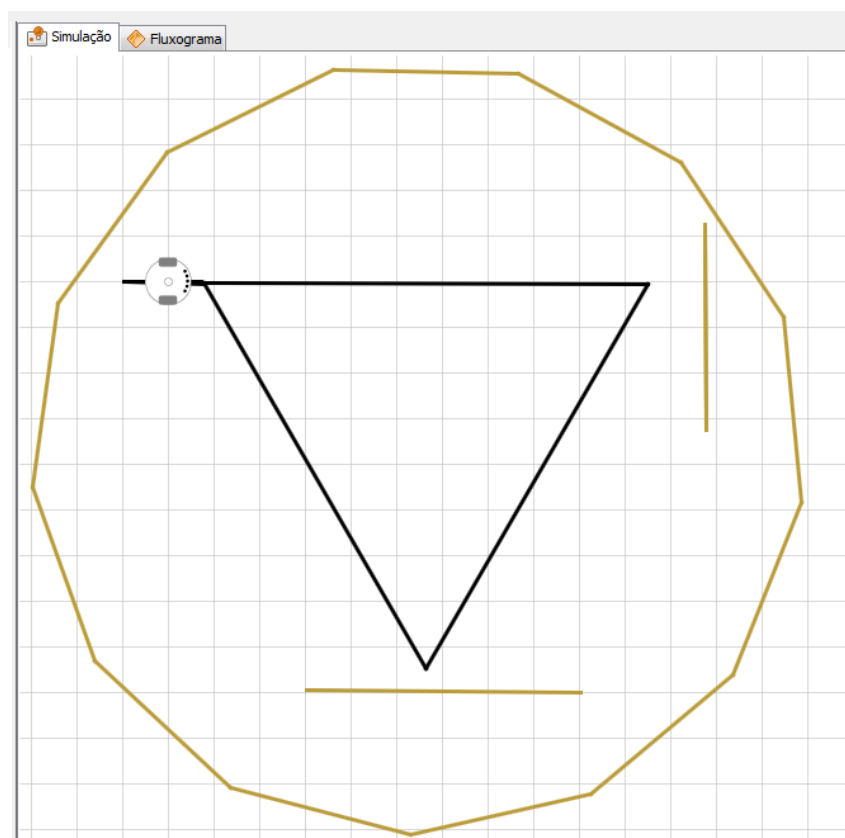
 **Linha Fechada** : Cria linhas no chão em forma de um polígono.

 **Parede** : Parede ou obstáculo, detectado pelo sensor de distância.


 **Parede Fechada** : Cria paredes em forma de um polígono.


 **Remover** : Remove elementos na intersecção com esta linha.


Exemplo aplicando os botões de desenvolvimento da aba simulação:





Além disso, na aba fluxograma os botões de desenvolvimento são especificamente blocos de comando, listados a seguir:


 **Procedimento** : Usado para declarar e atualizar o valor de variáveis, criando fórmulas e expressões algébricas.


 **Condicional** : Divide o fluxo em dois caminhos, um deles é escolhido pelo valor da condição fornecida.


 **Repetição** : Repete os comandos internos enquanto a condição fornecida for verdadeira.


 **Parar Repetição** : Interrompe o laço de execução quando é executado.


 **Ler Sensor** : Obtém o valor de um sensor e o armazena em uma variável.

 **Mover** : Envia uma mensagem para o robô mover as rodas, esquerda e direita, com velocidade v1 e v2 respectivamente. Valores diferentes fazem o robô girar ou andar para trás.

 **Girar** : Permite rotacionar o robô em ângulos bem definidos.

 **Esperar** : Mantém o robô se movendo ou faz o robô esperar por alguns milissegundos.

 **Exibir** : Quando executado exibe uma mensagem no Terminal, útil para saber o valor de variáveis em alguns pontos do programa.

 **Comentário** : Usado para deixar notas e dicas sobre o funcionamento do seu programa, não interfere na execução do mesmo.

Para adicionar blocos de comando ao programa principal, basta dois cliques. Primeiro para escolher o bloco e depois para soltar o bloco no local desejado. Depois do comando adicionado pode-se editá-lo clicando duas vezes em cima do bloco.

PENSAMENTO COMPUTACIONAL

Ao pensar em computação as pessoas comumente não a veem como algo diretamente dependente dos seres humanos. Costuma-se imaginar a programação de computadores como algo distante do mundo tangível, quase mágico, quando na verdade o cerne da programação está em qualquer atividade cotidiana: o pensamento computacional.

Pensar computacionalmente inclui ver as coisas de forma lógica e cadenciada, como um passo-a-passo para resolver um problema ou realizar uma tarefa (a isso podemos chamar de algoritmo), mas não se resume a isso. Inclui todas as áreas do conhecimento, humanas, exatas e biológicas.

A palavra-chave no pensamento computacional é abstração. A abstração na computação é feita de forma muito ampla, se lida não somente com números e respostas exatas, mas com situações reais e que a princípio parecem simplórias (um bom exemplo, e adiantando o que veremos mais à frente, como fazer um robô se movimentar sem bater nas pessoas ou obstáculos ao redor) se faz necessário pensar em situações limite e em todas as coisas que podem dar errado.

Outro conceito essencial no pensamento computacional é a automação. Isso significa mecanizar a abstração o máximo possível, deixando o “trabalho pesado” para as máquinas.

Relacionados diretamente com a automação e a abstração estão os algoritmos. Um algoritmo é uma sequência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema. Ele é composto por ações simples e bem definidas (não pode haver ambiguidade, ou seja, cada instrução representa uma ação que deve ser entendida e realizada). Exemplos de algoritmos variam de triviais, como receitas de culinária e manuais de instruções, à softwares que controlam trânsito de dados na web.

Na computação, algoritmos são geralmente ensinados em pseudocódigos, que representam a resolução de um problema usando estruturas de decisão típicas de linguagens de programação. Dentro desse raciocínio, é possível e desejável aglomerar em uma mesma estrutura vários comandos simples que estejam relacionados à mesma tarefa. A essas estruturas chamamos de funções.