// dia 18 com os navegar e centrar novos

// Imports necessᲩos

**import** com**.**ridgesoft**.**intellibrain**.**IntelliBrain**;**

**import** com**.**ridgesoft**.**io**.**Display**;**

// import com.ridgesoft.robotics.PushButton; // getDigitalIO

**import** com**.**ridgesoft**.**robotics**.**Motor**;**

**import** com**.**ridgesoft**.**robotics**.**ContinuousRotationServo**;**

**import** com**.**ridgesoft**.**robotics**.**AnalogInput**;**

**import** com**.**ridgesoft**.**robotics**.**RangeFinder**;**

**import** com**.**ridgesoft**.**robotics**.**sensors**.**ParallaxPing**;**

**import** com**.**ridgesoft**.**intellibrain**.**IntelliBrainDigitalIO**;**

public class RobBomb\_1 **{**

// Constantes para os estados da mᱵina

// Nota: mᱵina de estados nos apontamentos

private static final int ESPERAR **=** 0**;**

private static final int NAVEGAR\_E **=** 1**;**

private static final int CENTRAR **=** 2**;**

private static final int APAGAR **=** 3**;**

private static final int NAVEGAR\_D **=** 4**;**

// Constantes

// Nota: limites costumam ser os valores que necessitam de um ajuste quando o ambiente do robda (devido ࠩlumina磯, etc.)

// private static final int limiteLinha = 100; // Usada no sensor da linha. Compara-se o valor do sensor com o limite. Se for <, estᠥm cima do branco, linha = true. Se for >=, linha = false. Nota: se estiver em cima do preto, dᠹ00~1000; em cima do branco, dᠾ40. 100 頵m valor aceitᶥl.

// Para os sensores de chama, hᠤois tipos usados nas aulas:

//Tipo A: 500

//Tipo B: 700

private static final int limiteLinha **=** 100**;** // Usada no sensor da linha. Compara-se o valor do sensor com o limite. Se for <, estᠥm cima do branco, linha = true. Se for >=, linha = false. Nota: se estiver em cima do preto, dᠹ00~1000; em cima do branco, dᠾ40. 100 頵m valor aceitᶥl.

private static final int limiteChamaE **=** 450**;** // Com a vela acesa a uma dada dist⮣ia, dᠵm valor acima de 700. Variᶥl passe a true (chama presente) s󠱵ando o valor for superior a 700. Valor menor, considera-se que n㯠頣hama, mas radia磯 do ambiente.

private static final int limiteChamaD **=** 450**;** // Nota: necessᲩo afinar para rob> tipo A e B (fototransores n㯠s㯠todos iguais)

private static final int limiteChamaPerto **=** 990**;** // At頯s sensores atingirem este valor, o robo centra na chama

private static final int powerBase **=** 8**;** // Velocidade base

private static final int powerRodar **=** 5**;** // Velocidade para rodar sobre si pr󰲩o. Deverᠲodar devagar sobre si pr󰲩o

private static final float ganho **=** 0.27f**;** // Controlo proporcional: ver nomes de vars utilizadas na aula anterior

private static final int factorRodar **=** 10**;** // Para converter graus em tempo (m鴯do rodar()). Poderᠳer necessᲩo afina磯. 90 graus \* 10 milissegundos = 900 ms

// Objectos para os recursos de hardware

private static Display display**;**

private static Motor motorE**;**

private static Motor motorD**;**

private static Motor ventoinha**;**

private static AnalogInput sensorVelaE**;**

private static AnalogInput sensorVelaD**;**

private static AnalogInput sensorLinha**;**

private static IntelliBrainDigitalIO UVTron**;**

private static RangeFinder sonarE**;**

private static RangeFinder sonarF**;**

private static RangeFinder sonarD**;**

private static IntelliBrainDigitalIO botaoVerde**;**

private static IntelliBrainDigitalIO botaoVermelho**;**

private static IntelliBrainDigitalIO led\_verde**;**

private static IntelliBrainDigitalIO bumperD**;**

private static IntelliBrainDigitalIO bumperE**;**

// Distancias minimas das paredes

private static final int minDistF **=** 12**;**

private static final int minDistD **=** 13**;**

private static final int minDistE **=** 13**;**

public static void main**(**String**[]** args**)** **{**

// Estrutura tca da implementa磯 de uma mᱵina de estados

**try** **{**

//Cria磯 dos objectos

// Nota: Verificar liga絥s na placa

// LCD

display **=** IntelliBrain**.**getLcdDisplay**();**

display**.**print**(**0**,** "JEM b0t"**);**

// Motores

motorE **=** **new** ContinuousRotationServo**(**IntelliBrain**.**getServo**(**1**),** **false,** 14**);**

motorD **=** **new** ContinuousRotationServo**(**IntelliBrain**.**getServo**(**2**),** **true,** 14**);**

// Ventoa

ventoinha **=** IntelliBrain**.**getMotor**(**1**);**

// Sensores Chama

sensorVelaE **=** IntelliBrain**.**getAnalogInput**(**1**);** // Sensor chama esquerda

sensorVelaD **=** IntelliBrain**.**getAnalogInput**(**2**);** // Sensor chama direita

UVTron **=** IntelliBrain**.**getDigitalIO**(**6**);** //UVTron

//UVTron.enablePulseMeasurement(true);

// Sensor linha

sensorLinha **=** IntelliBrain**.**getAnalogInput**(**4**);**

// Sensores Sonar

sonarE **=** **new** ParallaxPing**(**IntelliBrain**.**getDigitalIO**(**3**));**

sonarF **=** **new** ParallaxPing**(**IntelliBrain**.**getDigitalIO**(**4**));**

sonarD **=** **new** ParallaxPing**(**IntelliBrain**.**getDigitalIO**(**5**));**

// Bot

botaoVerde **=** IntelliBrain**.**getDigitalIO**(**11**);**

botaoVerde**.**setPullUp**(true);** // Para o bot㯠funcionar, ligado ࠰orta 11 e GND. Liga o circuito atrav鳠de uma resistꮣia interna de +5V (para estabilizar circuito). Assim, tem referꮣia de tens㯬 sen㯠tem comportamento errᴩco.

botaoVermelho **=** IntelliBrain**.**getDigitalIO**(**12**);**

botaoVermelho**.**setPullUp**(true);**

// LED

led\_verde **=** IntelliBrain**.**getDigitalIO**(**10**);**

led\_verde**.**setDirection**(true);**

// Bumpers

bumperD **=** IntelliBrain**.**getDigitalIO**(**13**);**

bumperD**.**setPullUp**(true);**

bumperE **=** IntelliBrain**.**getDigitalIO**(**8**);**

bumperE**.**setPullUp**(true);**

// Variᶥis

boolean in\_quarto **=** **false;** // True se estᠮo quarto

boolean vela\_apagada **=** **false;** // Passa a true depois da vela apagada

boolean circulo **=** **false;** // Ponto de partida/chegada

int count **=** 0**;** // Tentativas para apagar a vela

int is\_chama**;** // Count edge UVTron

int distE**;**

int distF**;**

int distD**;**

int aux\_distF**;**

float erro**;** // Erro controlo proporcional

float delta**;** // Delta controlo proporcional

// Estado inicial

int estado **=** ESPERAR**;**

**while(true)** **{**

// Leitura dos sensores

// ɠtudo lido no mesmo so, mas no projecto poderᠨaver leituras localizadas

boolean linha**;**// = (sensorLinha.sample() < limiteLinha); // Para saber se estᠥm cima do preto ou branco (bool, que interessa para o algoritmo)

boolean apagarChamaE **=** **(**sensorVelaE**.**sample**()** **>** limiteChamaPerto**);**

boolean apagarChamaD **=** **(**sensorVelaD**.**sample**()** **>** limiteChamaPerto**);**

// Inicia o UVTRon

//UVTron.enablePulseMeasurement(true);

// Sonares

// Nota: getDistance devolve -1 quando o som se perde. Com arredondamento de +0.5f para o inteiro mais pr󸩭o, nunca iria dar.

// Teste sensores --> Tirar comentᲩo ao valor para testar

// Pstado ESPERAR comentado no switch

// Nota: 1 de cada vez

// Sensor de linha

// mostrarValores(sensorLinha.sample(), 0); // Funciona como esperado

// Sensores de chama

//mostrarValores(sensorVelaE.sample(), sensorVelaD.sample()); // S㯠os dois do mesmo tipo

//UVTron.enablePulseMeasurement(true);

//Thread.sleep(2000);

//mostrarValores(UVTron.readEdgeCount(),(UVTron.readEdgeCount()>=50?1:0));

//Thread.sleep(1000);

//UVTron.enablePulseMeasurement(false);

// Dist⮣ias dos sensores

// mostrarValores(distE, distF); // Funcionam como esperado

// Booleanos

// mostrarValores(linha?1:0, 0); // Funciona como esperado

// Bot㯍

// mostrarValores(botaoVerde.isSet()?1:0, 0); // Funciona como esperado

// Bumpers

//int bumper\_esq = 0;

//int bumper\_dir = 0;

//if (!bumperEsquerda.isSet())

//bumper\_esq = 1;

//if (!bumperDireita.isSet())

//bumper\_dir = 1;

//mostrarValores(bumper\_esq, bumper\_dir);

// Se bot㯠vermelho carregado fica em espera

**if(!**botaoVermelho**.**isSet**())** **{**

estado **=** ESPERAR**;**

**}**

// Switch para mudar de estado

///\*

**switch** **(**estado**)** **{**

**case** ESPERAR**:**

// Comportamento propriamente dito

mostrarEstado**(**0**);**

parar**();**

in\_quarto **=** **false;**

vela\_apagada **=** **false;**

circulo **=** **false;**

count **=** 0**;**

is\_chama **=** 0**;**

erro **=** 0.0f**;**

delta **=** 0.0f**;**

**while(**botaoVerde**.**isSet**())**

**;**

// Condi絥s

avancar**(**4**);**

Thread**.**sleep**(**2000**);**

estado **=** NAVEGAR\_E**;**

**break;**

**case** NAVEGAR\_E**:**

mostrarEstado**(**1**);**

linha **=** sensorLinha**.**sample**()** **<** limiteLinha**;**

// Comportamento

sonarE**.**ping**();**

Thread**.**sleep**(**30**);**

distE **=** **(**int**)** **(**sonarE**.**getDistanceCm**());**

sonarF**.**ping**();**

Thread**.**sleep**(**30**);**

distF **=** **(**int**)** **(**sonarF**.**getDistanceCm**());**

navegaEsq**(**distE**,** distF**);** // Navega bem

// Condi絥s

// Bumpers

**if** **(!**bumperE**.**isSet**()){**

avancar**(-**4**);**

Thread**.**sleep**(**1000**);**

rodar**(-**50**);**

**}**

**if** **(!**bumperD**.**isSet**())** **{**

avancar**(-**4**);**

Thread**.**sleep**(**1000**);**

rodar**(**50**);**

**}**

// Se estiver perto da parede roda 75, para evitar bater

**if** **(**distF **<** minDistF**)** **{**

rodar**(-**35**);**

**}**

// Se estiver perto da parede roda 75, para evitar bater

**if** **(**distE **<** minDistE**)** **{**

rodar**(-**35**);**

**}**

// Se detecta linha branca e for de um quarto > liga UVTron

**if** **(**linha**)** **{**

//parar();

avancar**(**2**);**

Thread**.**sleep**(**750**);**

in\_quarto **=** **(**sensorLinha**.**sample**()** **>** limiteLinha**);**

**if** **(**in\_quarto **&&** **!**vela\_apagada**)** **{**

// Espera 2 segundos para a leitura do UVTron

UVTron**.**enablePulseMeasurement**(true);**

Thread**.**sleep**(**2000**);**

is\_chama **=** UVTron**.**readEdgeCount**();**

// Pontador a 0

UVTron**.**enablePulseMeasurement**(false);**

// Se passa limite -> presen硠chama e passa ao centrar

**if** **(**is\_chama **>=** 20**)** **{**

led\_verde**.**set**();**

estado **=** CENTRAR**;**

**}** **else** **{**

// Se n㯠detecta chama no quarto roda e avan硠um bocado e volta a navegar

avancar**(-**5**);**

Thread**.**sleep**(**1500**);**

rodar**(-**170**);**

estado **=** NAVEGAR\_E**;**

**}** // if is\_chama

**}** **else** **if** **(!**vela\_apagada**)** **{** // if !in\_quarto

avancar**(**4**);**

Thread**.**sleep**(**2000**);**

sonarF**.**ping**();**

Thread**.**sleep**(**30**);**

aux\_distF **=** **(**int**)** **(**sonarF**.**getDistanceCm**());**

**if** **(**aux\_distF **<** 15**)** **{**

rodar**(-**90**);**

sonarF**.**ping**();**

Thread**.**sleep**(**30**);**

aux\_distF **=** **(**int**)** **(**sonarF**.**getDistanceCm**());**

**if** **(**aux\_distF **<** 15**)** **{**

rodar**(-**90**);**

rodar**(-**90**);**

**}**

**}**

**}**

**}** // if linha

**break;**

**case** NAVEGAR\_D**:**

mostrarEstado**(**4**);**

// Comportamento

sonarD**.**ping**();**

Thread**.**sleep**(**30**);**

distD **=** **(**int**)** **(**sonarD**.**getDistanceCm**());**

sonarF**.**ping**();**

Thread**.**sleep**(**30**);**

distF **=** **(**int**)** **(**sonarF**.**getDistanceCm**());**

navegaDir**(**distD**,** distF**);**

// Bumpers

**if** **(!**bumperE**.**isSet**()){**

avancar**(-**4**);**

Thread**.**sleep**(**1000**);**

rodar**(-**50**);**

**}**

**if** **(!**bumperD**.**isSet**())** **{**

avancar**(-**4**);**

Thread**.**sleep**(**1000**);**

rodar**(**50**);**

**}**

// Se estiver perto da parede roda 75, para evitar bater

**if** **(**distF **<** minDistF**)** **{**

rodar**(-**35**);**

**}**

// Se estiver perto da parede roda 75, para evitar bater

**if** **(**distD **<** minDistD**)** **{**

rodar**(**35**);**

**}**

// Condi絥s

//if(se for circulo) estado = esperar

linha **=** sensorLinha**.**sample**()** **<** limiteLinha**;**

**if** **(**linha**)** **{**

**if** **(**in\_quarto**)** **{**

in\_quarto **=** **false;**

**}** **else** **{**

avancar**(**3**);**

Thread**.**sleep**(**750**);**

circulo **=** **(**sensorLinha**.**sample**()** **<** limiteLinha**);**

**if** **(**circulo**)** **{**

parar**();**

estado **=** ESPERAR**;**

**}** **else** **{**

avancar**(-**4**);**

Thread**.**sleep**(**1500**);**

rodar**(**130**);**

**}**

**}** // else !in\_quarto

**}** // if linha

**break;**

**case** CENTRAR**:**

mostrarEstado**(**2**);**

// Comportamento

// sensorE - x < sensorD < sensorE + x

// OU

// sensorD - x < sensorE < sensorD + x

**if** **(** **((**sensorVelaE**.**sample**()** **-** 20**)** **<** sensorVelaD**.**sample**())** **&&** **((**sensorVelaE**.**sample**()** **+** 20**)** **>** sensorVelaD**.**sample**())** **)** **{** // Se vela ࠦrente, avan硍

avancar**(**5**);**

mostrarValores**(**1**,**0**);**

**}** **else** **if** **(**sensorVelaE**.**sample**()** **<** sensorVelaD**.**sample**())** **{** // Se vela ࠥsquerda, vira ࠥsquerda

arco**(**5**,** **-**3**);**

mostrarValores**(**2**,**0**);**

**}** **else** **if** **(**sensorVelaE**.**sample**()** **>** sensorVelaD**.**sample**())** **{** // Se vela ࠤireita, vira ࠤireita

arco**(**5**,** 3**);**

mostrarValores**(**3**,**0**);**

**}** **else** **{**

// Se perder a vela, procura

**while** **(** **(**sensorVelaE**.**sample**()** **<** limiteChamaE**)** **||** **(**sensorVelaD**.**sample**()** **<** limiteChamaD**))** **{**

rodar**(**10**);**

**}**

**}**

// Condi絥s

// Apaga quando a chama estiver perto

**if** **(**apagarChamaE **&&** apagarChamaD**)** **{**

avancar**(**2**);**

Thread**.**sleep**(**1000**);**

parar**();**

estado **=** APAGAR**;**

mostrarValores**(**5**,** 0**);**

**}**

mostrarValores**(**6**,** 0**);**

**break;**

**case** APAGAR**:**

mostrarEstado**(**3**);**

// Comportamento

// Fun磯 para ligar a ventoa

apagarVela**();**

UVTron**.**enablePulseMeasurement**(true);**

// Espera um bocado para verificar se a vela ainda estᠡcesa

Thread**.**sleep**(**5000**);**

// Condi絥s

// Espera 2 segundos para a leitura do UVTron

//UVTron.enablePulseMeasurement(true);

//Thread.sleep(2000);

is\_chama **=** UVTron**.**readEdgeCount**();**

// Pontador a 0

UVTron**.**enablePulseMeasurement**(false);**

// Se passa limite -> vela continua acesa

**if** **(** **(**is\_chama **>=** 20**)** **&&** **(**count **<=** 2**)** **)** **{**

// Incrementa as tentativas para apagar

**++**count**;**

**}** **else** **if** **(** **(**is\_chama **>=** 20**)** **&&** **(**count **>=** 2**)** **)** **{** // Se a vela continuar acesa e o nmero de tentativas for >= 2

estado **=** CENTRAR**;**

**}** **else** **{** // Sen㯬 a vela estᠡpagada e regressa ao estado navegar para voltar a casa

rodar**(**90**);**

vela\_apagada **=** **true;** // Vela foi apagada

led\_verde**.**clear**();**

estado **=** NAVEGAR\_D**;**

**}**

**break;**

**}** // Fim switch \*/

**}** // Fim while

**}** // Fim try

**catch** **(**Throwable t**)** **{**

t**.**printStackTrace**();**

**}** // Fim catch

**}** // Fim main

// Fun絥s adicionais

// Faz arco

public static void arco**(**int power**,** int factor**)** **{**

motorE**.**setPower**(**power **+** factor**);**

motorD**.**setPower**(**power **-** factor**);**

**}** // Fim arco

// Avan硠simplesmente com o power fornecido

public static void avancar**(**int power**)** **{**

motorE**.**setPower**(**power**);**

motorD**.**setPower**(**power**);**

**}** // Fim avancar

// Escreve os inteiros que forem passados como par⭥tros --> DEBUG

// Por exemplo, verificar se os sensores est㯠a funcionar

public static void mostrarValores**(**int v1**,** int v2**)** **{**

display**.**print**(**0**,** Integer**.**toString**(**v1**));**

display**.**print**(**1**,** Integer**.**toString**(**v2**));**

**}** // Fim mostrarValores

// Mostra o estado do rob LCD

public static void mostrarEstado**(**int estado**)** **{**

**switch** **(**estado**)** **{**

**case** ESPERAR**:**

display**.**print**(**1**,** "ESPERAR"**);**

**break;**

**case** NAVEGAR\_E**:**

display**.**print**(**1**,** "NAVEGAR\_E"**);**

**break;**

**case** CENTRAR**:**

display**.**print**(**1**,** "CENTRAR"**);**

**break;**

**case** APAGAR**:**

display**.**print**(**1**,** "APAGAR"**);**

**break;**

**case** NAVEGAR\_D**:**

display**.**print**(**1**,** "NAVEGAR\_D"**);**

**break;**

**}** // Fim switch

**}** // Fim mostrarEstado

// Roda sobre si pr󰲩o x graus

public static void rodar**(**int graus**)** **{**

**if** **(**graus **<** 0**)** **{**

graus **=** **-**graus**;**

motorE**.**setPower**(**powerRodar**);**

motorD**.**setPower**(-**powerRodar**);**

**}** **else** **{**

motorE**.**setPower**(-**powerRodar**);**

motorD**.**setPower**(**powerRodar**);**

**}**

**try** **{**

Thread**.**sleep**(**graus **\*** factorRodar**);** // Graus convertidos para tempo (3simples)

**}**

**catch** **(**Throwable t**)** **{**

t**.**printStackTrace**();**

**}**

// PᲡ o rob􍊼/span> parar**();**

**}** // Fim rodar

// P potꮣia dos motores a 0

public static void parar**()** **{**

motorE**.**setPower**(**0**);**

motorD**.**setPower**(**0**);**

**}** // Fim parar

// Robda para apagar

public static void apagarVela**()** **{**

**try** **{**

ventoinha**.**setPower**(**16**);**

Thread**.**sleep**(**5000**);**

ventoinha**.**setPower**(**0**);**

**}**

**catch** **(**Throwable t**)** **{**

t**.**printStackTrace**();**

**}**

**}** // Fim apagarVela

// Navega seguindo a parede com o sonar da esquerda e corrige trajecto

public static void navegaEsq**(**int distE**,** int distF**)** **{**

**try** **{**

// Para o sensor da frente

**if** **(**distF **<** 30**)** **{**

motorE**.**setPower**(**14**);**

motorD**.**setPower**(**3**);**

Thread**.**sleep**(**600**);**

**}**

// Com controlo proporcional

// Guarda o erro

// Considerou-se D = 15 cm

float erro **=** distE **-** 17.0f**;**

// Para evitar comportamentos indesejados...

// Quando o erro > 15, causa comportamentos indesejados.

// Acontece quando o sensor aponta para uma dist⮣ia maior

// ao corrigir o trajecto (ver apontamentos)

**if** **(**erro **>** 17**)**

erro **=** 17.0f**;**

// Calcula o delta

float delta **=** erro **\*** ganho**;**

// Aplica potꮣia consoante o erro

motorE**.**setPower**(** **(**int**)(**powerBase **-** delta**)** **);**

motorD**.**setPower**(** **(**int**)(**powerBase **+** delta**)** **);**

**}** // Fim try

**catch** **(**Throwable t**)** **{**

t**.**printStackTrace**();**

**}** // Fim catch

**}** // Fim navegaEsq

// Navega seguindo a parede com o sonar da direita e corrige trajecto

public static void navegaDir**(**int distD**,** int distF**)** **{**

**try** **{**

// Para o sensor da frente

**if** **(**distF **<** 30**)** **{**

motorE**.**setPower**(**3**);**

motorD**.**setPower**(**14**);**

Thread**.**sleep**(**600**);**

**}**

// Com controlo proporcional

// Guarda o erro

// Considerou-se D = 15 cm

float erro **=** distD **-** 17.0f**;**

// Para evitar comportamentos indesejados...

// Quando o erro > 15, causa comportamentos indesejados.

// Acontece quando o sensor aponta para uma dist⮣ia maior

// ao corrigir o trajecto (ver apontamentos)

**if** **(**erro **>** 17**)**

erro **=** 17.0f**;**

// Calcula o delta

float delta **=** erro **\*** ganho**;**

// Aplica potꮣia consoante o erro

motorE**.**setPower**(** **(**int**)(**powerBase **+** delta**)** **);**

motorD**.**setPower**(** **(**int**)(**powerBase **-** delta**)** **);**

**}** // Fim try

**catch** **(**Throwable t**)** **{**

t**.**printStackTrace**();**

**}** // Fim catch

**}** // Fim navegaDir

**}** // Fim RobBomb\_1