DDETC – Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

MEIM - Mestrado Engenharia informática e multimédia

Robótica Móvel

Trabalho prático 1

Turma:

MEIM-2D

Trabalho realizado por:

Miguel Távora Nº45102

Bruno Colaço Nº45037

Docente:

Jorge Pais

Data: 22/02/2022

Índice

Índice ilustrações e tabelas

1. Introdução

A robótica é um ramo tecnológico que trata de sistemas compostos por partes mecânicas em conjunto com circuitos integrados. Os sistemas mecânicos motorizados são controlados por circuitos elétricos e por uma inteligência computacional.

Cada vez mais as pessoas utilizam os robôs para as suas tarefas, por exemplo: o robô aspirador, robôs para cirurgias médicas entre outros. A utilização da robótica em geral introduziu redução de custos, aumento da produtividade e redução de erros.

Um robô é um dispositivo ou grupo de dispositivos capazes de realizar trabalhos de maneira autônoma ou pré-programada. Um robô é então constituído por diversas partes nomeadamente:

- Mecânica: composto por motores, rodas, servos, etc.
- Eletrónica: circuitos, sensores, etc.
- Sensores: áudio, imagem, temperatura, etc.
- Controlo: modelos matemáticos, modelos cinemáticos, etc.
- Algoritmos: modelos estruturados descritos em diagramas de atividade abstratos implementados numa linguagem de programação.
- Computador: o “cérebro” das máquinas artificiais criados pelo homem.

Robôs automáticos e famosos são por exemplo:

- Curiosity – é um robô projeto para explorar a cratera Gale em Marte como projeto da NASA. Os objetivos desta sonda incluem investigação do clima e da geologia marciana.
- Plustech – primeiro robô industrial capaz de apanhar troncos de madeira numa floresta.
- Pionner – robô construído para desmantelar os destroços do reator destruído, entre outras funções. Para isso ser possível é necessário que o robô consiga aguentar em condições de alto nível de radiação.

- NXT e EV3 – robôs modulares fabricados pela Lego de forma a aprender robótica.

1.1 Objetivo

Este trabalho tem como objetivo o desenvolvimento de uma interface gráfica para controlar o robô Lego EV3. Além da implementação da interface gráfica será também desenvolvido uma implementação feita pelo grupo da biblioteca robotLegoEV3.jar, biblioteca esta que permite realizar o andamento do robô em linha reta e em curvas. Através da implementação será possível consolidar os conhecimentos relacionados com o controlo e os algoritmos da robótica.

2. Desenvolvimento

1. Interface gráfica

A interface gráfica é um processo local que é responsável por enviar comandos para o robô Lego EV3. Para desenvolver esta interface gráfica foi utilizado o editor de interfaces gráficas para o Java, Window Builder. As classes utilizadas para construir a interface gráfica encontram-se na pasta Swing do Java, onde através da instância de classes como JFrames, JButtons, entre outros é possível construir a interface pretendida.

A interface gráfica possui a seguinte representação:

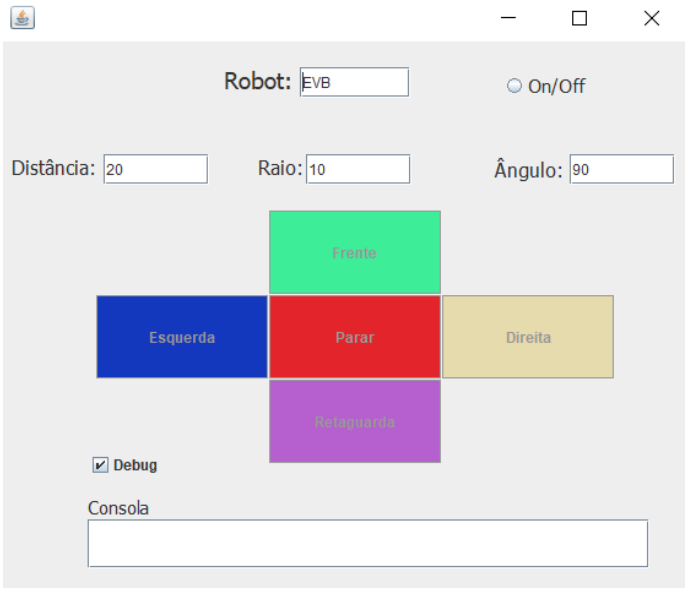

Figura 1 - Interface gráfica de controlo do robô Lego EV3

A interface é constituída por diversos elementos responsáveis por enviar comandos, definir valores e para *debug*. Os elementos são nomeadamente:

- Botão Frente – envia um comando para o robô andar em linha reta para a frente a distância especificada no campo Distância, este valor está em centímetros.
- Botão Retaguarda – realiza o movimento em linha reta para trás com o valor

especificado no campo da Distância.

- Botão Esquerda – envia um comando para o robô curvar para a esquerda, onde a curva possui o raio especificado no campo Raio e o ângulo do campo Ângulo.
- Botão Direita – envia um comando para o robô curvar para a direita, onde a curva possui o raio especificado no campo Raio e o ângulo do campo Ângulo.
- Botão Parar – manda o robô parar no final da realização de um comando. Caso não seja clicado neste botão o robô fica eternamente a realizar o último comando enviado para ele.
- Botão On/Off – responsável por ligar e desligar a comunicação Bluetooth com o robô. Quando se clica ele conecta-se com o robô e caso tenha conseguido ligar fica ativo, caso contrário continua desselecionado. Caso o robô esteja ligado e se clique neste botão ele desconecta-se com o robô. Os botões só ficam ativos após o computador ter conseguido conectar-se com sucesso com o robô.
- Campo Robot – onde se define o nome do robô ao qual se vai conectar. Após escrever o nome é necessário clicar Enter para assumir o comando.
- Campo Distância – distância que o robô irá percorrer quando é selecionado o botão Frente ou Retaguarda. O valor da distância é em centímetros e na altura da inserção é necessário clicar Enter para assumir o novo valor. Caso na inserção tenham sido inseridas letras é envia uma mensagem de aviso que só aceitas números e os números têm de ser inferiores a 401.
- Campo Raio – raio do círculo feito pela curva quando se clica no botão Esquerda ou Direita. Da mesma maneira que o anterior valida o conteúdo da mensagem e o valor tem de ser inferior a 51.
- Campo Ângulo – ângulo em graus percorrido pelo robô numa curva através do clique no botão Esquerda ou Direita. Da mesma maneira que os dois anteriores valida o conteúdo e o seu valor tem de ser inferior a 721.
- Checkbox Debug – quando esta *checkbox* está ativa imprime na consola os comandos que estão a ser enviados para o robô. A consola não é possível escrever, pois só serve como *debugger*.

- Botão X – na altura que é terminada a interface gráfica o robô ainda pode estar conectado, gerando problemas pois o canal de comunicação fica ocupado e não é possível enviar mais nenhum comando para o robô. Sendo assim necessário estar sempre a desligar e ligar o robô. Para resolver isso na altura de terminar é sempre terminada a conexão com o robô.

2. Implementação da biblioteca robotLegoEV3.jar

O objetivo principal do trabalho é desenvolver conhecimentos na área da robótica. Para isso será desenvolvida uma implementação da biblioteca robotLegoEV3.jar feita pelo grupo. A biblioteca do robotLegoEV3 possui 6 métodos distintos. Estes métodos são nomeadamente:

- CurvarEsquerda – realiza uma curva para a esquerda com um determinado raio e um ângulo passados como argumento.
- CurvarDireita – realiza uma curva para a direita com um determinado raio e um ângulo passados como argumento.
- Reta – realiza uma linha reta com o comprimento passado como argumento em centímetros.
- Parar – método que recebe como argumento um booleano. Caso seja verdade (*true*) ele pára imediatamente o robô mesmo estando a meio de um comando, caso seja falso (false) ele pára o robô após cumprir o último comando enviado para o mesmo.
- OpenEV3 - método para estabelecer o canal de comunicação do robô com o computador. Para isso ele recebe o nome do robô e tenta estabelecer a comunicação e retorna um booleano a verdade quando consegue estabelecer o canal e falso caso contrário.
- CloseEV3 – método para fechar o canal de comunicação.

3. Implementação da classe de controlo do robô Lego EV3

3.1 Instalação dos programas

Primeiramente para a produção do sistema foi utilizado a linguagem de programação Java, do Java foi utilizado o Java Runtime Enviroment (JRE) de forma a conseguir adicionar bibliotecas hardware externas ao próprio Java. Para adicionar bibliotecas externas foi necessário ir à pasta onde ficou instalado o Java, dentro da pasta lib ir à pasta ext e adicionar as bibliotecas dentro dessa pasta. Após isto foi também necessário configurar o editor para que o compilador (javac) fosse o mesmo que executa o código(java).

3.2 Instalação e descrição da biblioteca InterpretadorEV3

Numa primeira fase da implementação foi somente utilizada a biblioteca do robotLegoEV3.jar realizada pelo docente, de maneira a conseguir testar se a instalação dos programas e das bibliotecas foi ou não feita corretamente. Esta biblioteca será também utilizada como termo de comparação dos resultados obtidos na nossa implementação.

Para criamos a nossa implementação foi utilizada uma biblioteca disponibilizada pelo docente para acesso aos componentes de hardware do robô designada InterpretadorEV3.jar. Esta biblioteca permite conectar e desconectar o robô com o computador e enviar comando para o mesmo, como por exemplo por as rodas em andamento. Toda esta comunicação é feita por Bluetooth, como esta comunicação é feita ponto a ponto é necessário estabelecer sempre um canal de comunicação entre os participantes. Sempre que já não se pretende enviar mensagens este canal deve ser fechado.

3.3 Descrição do robô Lego EV3

Existem diversas formas de locomoção, os sistemas biológicos geralmente movem-se com movimento de pernas ou patas. No entanto nos sistemas artificiais a locomoção é baseada em rodas porque a construção e controlo destes é mais simples. O robô Lego EV3 utiliza locomoção baseado em rodas, onde possui duas rodas frontais paralelas com movimento diferencial e uma roda traseira esférica. Cada motor tem associado um tacómetro que permite medir a rotação das rodas em graus com precisão de um grau.

Figura 2 - desenho do robô Lego EV3

3.4 Implementação da classe MyRobotLego

Para a criação do nosso controlador do robô Lego EV3, feito pela biblioteca robotLegoEV3.jar, foi criada uma classe designada MyRobotLego. Para a ser possível a utilização do robô foi instanciada a classe InterpretadorEV3, que possui diversos métodos para controlar o robô.

3.4.1 Implementação do método openEV3()

Nesta classe foi criado o método openEV3(), este método recebe como argumento o nome do robô e utiliza o método OpenEV3() do InterpretadorEV3. O nome do robô irá corresponder ao nome escrito no campo do robô da interface gráfica. Este método é responsável por estabelecer um canal de comunicação por meio do Bluetooth do computador com o robô e retorna um booleano com o valor verdade quando consegue estabelecer comunicação e falso caso contrário.

3.4.2 Implementação do método closeEV3()

Foi definido também o método closeEV3(), este método é responsável por terminar o canal de comunicação entre o computador e o robô. Este método antes de terminar a comunicação manda um comando para reiniciar todos valores do robô e parar os motores do mesmo, para prevenir que o robô fique por exemplo em andamento mesmo após terminar a comunicação. Por fim termina o canal de comunicação.

3.4.3 Implementação do método line()

Para o robô andar em linha reta uma determinada distância foi criado o método line() que recebe como argumento a distância que será percorrida pelo robô. Quando a distância é positiva o robô anda para a frente e quando é negativa ele anda para trás.

Para conseguir por o robô a andar é utilizado o comando OnFwd() da biblioteca do InterpretadorEV3, onde foi utilizado a mesma velocidade tanto para a roda esquerda como para a roda direita. Para realizar o andamento para trás do robô é utilizado a velocidade com um valor negativo. Para se saber quando é que o robô cumpriu a distância em centímetros recebida no argumento do método é necessário saber a quantidade de rotações feitas pelo motor do robô. Esta informação é obtida através do método RotationCount() do interpretador. Para conseguir a quantidade de rotações totais é utilizada a seguinte conta:

$$\hat{A}ngulo\ total = \frac{\left(\frac{distância}{raio\ das\ rodas}\right) * 180}{\pi}$$

Na fórmula anterior a distância corresponde à distância que é pretendido o robô percorrer. O raio das rodas é o raio das rodas do robô. A multiplicação por 180 e dividir pelo π é para converter o ângulo para graus. Através do método RotationCount() é possível obter as rotações feitas pelo robô em determinados instantes temporais. Quando o RotationCount() possuir um valor superior ao ângulo total é porque o robô já cumpriu a distância pretendida e deve terminar o autómato. Para evitar estar sistematicamente a realizar pedidos sobre a quantidade de rotações e provocar um sobre carregamento da memória do robô é feita uma espera com um valor arbitrário. Quando o robô anda para trás o valor das rotações é negativo e por isso é

utilizado uma variável designada *signal* para multiplicar com o valor das rotações convertendo sempre as rotações para um valor positivo.

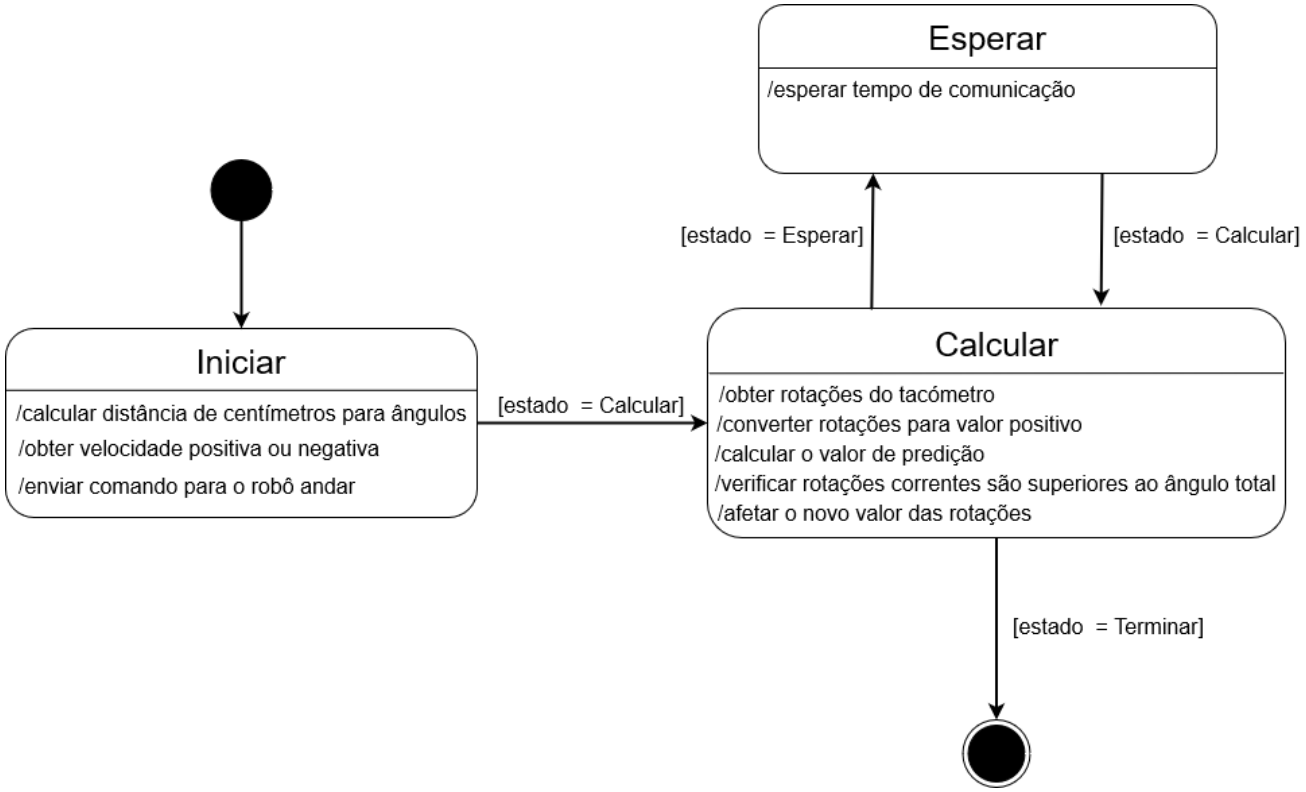

Figura 3 - diagrama de estados do método line()

Pelo diagrama de estados é possível observar então que inicialmente o robô começa as suas tarefas no estado Iniciar.

- Estado Iniciar - neste estado é feito o cálculo da distância em rotações que os motores das rodas devem fazer. Também neste estado é verificado se a distância é positiva ou negativa, visto que o andamento para trás do robô é feito através OnFwd() com velocidade negativa. Também neste estado é enviado o comando para pôr o robô a andar. Após acabar o estado Iniciar ele vai sempre para o estado Calcular.
- Estado Calcular – neste estado é chamado o método RotationCount() para obter a quantidade de rotações feitas num determinado instante que o robô está a andar. Quando o robô anda para trás o valor das rotações obtido é negativo, por isso o valor é sempre convertido para positivo. Para reduzir o erro é utilizada uma variável iniciada a zero para prever com maior exatidão onde o robô deve parar. Para isso no final do estado Calcular é sempre afetado o valor corrente do RotationCount(), este valor serve para

depois subtrair na próxima vez que correr o estado Calcular. O cálculo é feito a partir da diferença entre as rotações correntes e anteriores, obtendo um valor positivo e este valor é somado ao valor das rotações corrente. Por fim é verificado se as rotações correntes mais o valor de predição são superiores ao ângulo total, se sim o estado passa a ser Terminar senão é afetado como Esperar.

- Estado Esperar – este estado é responsável por realizar uma espera arbitraria, neste caso 200 milissegundos. Após completar a espera o autómato transita sempre para o estado Calcular.

3.4.3.1 Erro associado ao movimento

Contudo existe um problema que é a grande quantidade de erro associado ao movimento, onde por exemplo para uma reta de 20 centímetros ele realiza por vezes 26 centímetros. Isto deve-se a dois fatores principais: o truncamento dos valores e o tempo de comunicação. Para reduzir o erro em vez de se truncar os valores é feito um arredondamento. Como a implementação deste modelo é baseado no modelo Robô – Computador existe um atraso na receção da informação vinda do robô resultante do tempo de comunicação. Por isso o computador recebe sempre a informação atrasada no tempo, tomando decisões não no tempo presente, mas sim no tempo passado produzindo um erro maior nos resultados.

Para resolver este problema é utilizado uma técnica de predição, neste caso a técnica consistem em guardar as rotações obtidas no pedido anterior e calcular a diferença com as rotações atuais. No final é adicionada a diferença entre as rotações às rotações atuais obtidas. Desta forma é possível minimizar o erro ao máximo e obter erros muito inferiores aos erros produzidos sem as técnicas de predição. Isto pode-se observar na tabela que se segue:

Distância em cm	Distância percorrida (com técnica de predição)	Distância percorrida (sem técnica de predição)
0	(-) 6cm	(-) 5 cm
1	6 cm	6 - 7cm
5	6 - 7 cm	6 - 7cm
10	7.5 - 12 cm	11 - 13 cm
15	12.5 - 15	17.5 - 18 cm
20	18 - 20 cm	23 - 26 cm
30	26 -27 cm	32 - 35 cm
40	38 - 40.5 cm	43 cm - 46 cm
50	47 - 50.5 cm	56 - 57 cm

Tabela 1 - erro associado ao método line()

Para a realização destes testes foi utilizado sempre velocidade de 50 e o robô utilizado foi sempre o EVB. Perante os resultados observa-se que o erro máximo utilizando técnicas de predição está sempre entre menos 3 e mais 2 centímetros. Por outro lado, sem as técnicas de predição o valor do erro sobe substancialmente, sendo o erro entre mais 3 e mais 6 centímetros. Por isso utilizar a técnica de predição é benéfico porque é possível reduzir muito a quantidade de erro.

3.4.4 Implementação do método curve()

Para o robô realizar uma curva com um determinado raio e um determinado ângulo foi implementado o método curve(), este método recebe como argumento se é uma curva para a direita ou esquerda, o ângulo e o raio. Quando o valor do primeiro argumento é verdade ele realiza uma curva para a direita, se for falso vai para a esquerda.

Para ser possível por o robô a curvar, da mesma forma que o método line(), foi utilizado o comando OnFwd() da biblioteca InterpretadorEV3. Contudo para realizar uma curva foi necessário por os motores das rodas a andar a uma velocidade diferente.

Quando se curva para a direita a roda esquerda deve ser a mais rápida e quando se curva para a esquerda deve ser a roda direita. Os valores da velocidade obtidos devem ser iguais aos

valores do ficheiro da manobrabilidade, onde no caso do trabalho foi escolhido velocidade de 50 e por isso a velocidade da roda mais rápida deve ser 73 e a mais lenta deve ser 26.

De igual forma que o método line(), também é verificado se o robô já cumpriu o ângulo pretendido. Para isso é necessário saber a quantidade de rotações feitas pelos motores do robô, obtido utilizado o método RotationCount() do interpretador.

3.4.4.1 Implementação das contas do curvar à direita

Para saber o número de rotações é necessário realizar alguns cálculos. Neste caso serão exibidos os cálculos para o curvar à direita, mais à frente serão exibidos os cálculos do curvar à esquerda.

Primeiramente é necessário calcular a distância entre rodas, o valor medido foi 9.5 centímetros. Para calcular a distância desde o centro do raio do círculo até à roda direita é dado pela fórmula:

$$Distância_{Roda\ direita} = Raio\ círculo - \frac{Distância\ entre\ rodas}{2}$$

$$Distância_{Roda\ esquerda} = Raio\ círculo + \frac{Distância\ entre\ rodas}{2}$$

A partir das distâncias calculadas anteriormente é possível calcular o fator entre a velocidade da roda esquerda e direita. O fator é dado por:

$$fator = \frac{Distância_{Roda\ esquerda}}{Distância_{Roda\ direita}}$$

A velocidade da roda direita é então dada pela fórmula:

$$Velocidade_{roda\ direita} = \frac{Velocidade\ robô * 2}{1 + fator}$$

Na fórmula anterior a velocidade do robô, como foi referido anterior, é 50. A partir desta fórmula a velocidade da roda esquerda vai ser então:

$$Velocidade_{roda\ esquerda} = Velocidade_{roda\ direita} * factor$$

Os valores obtidos nas fórmulas podem produzir números decimais, contudo a velocidade enviada para o robô tem sempre de ser um número inteiro. Neste sentido pode-se então arredondar ou truncar os valores. Nas contas foi sempre utilizado o arredondamento da velocidade, isto porque quando se trunca os valores os resultados obtidos possuem uma margem de erro superior.

Para calcular a quantidade de rotações totais para realizar a curva é necessário realizar a seguinte conta:

$$\hat{A}ngulo_{total} = \frac{\hat{a}ngulo_{pretendido} * raio_{ciruclo}}{Raio_{rodas}}$$

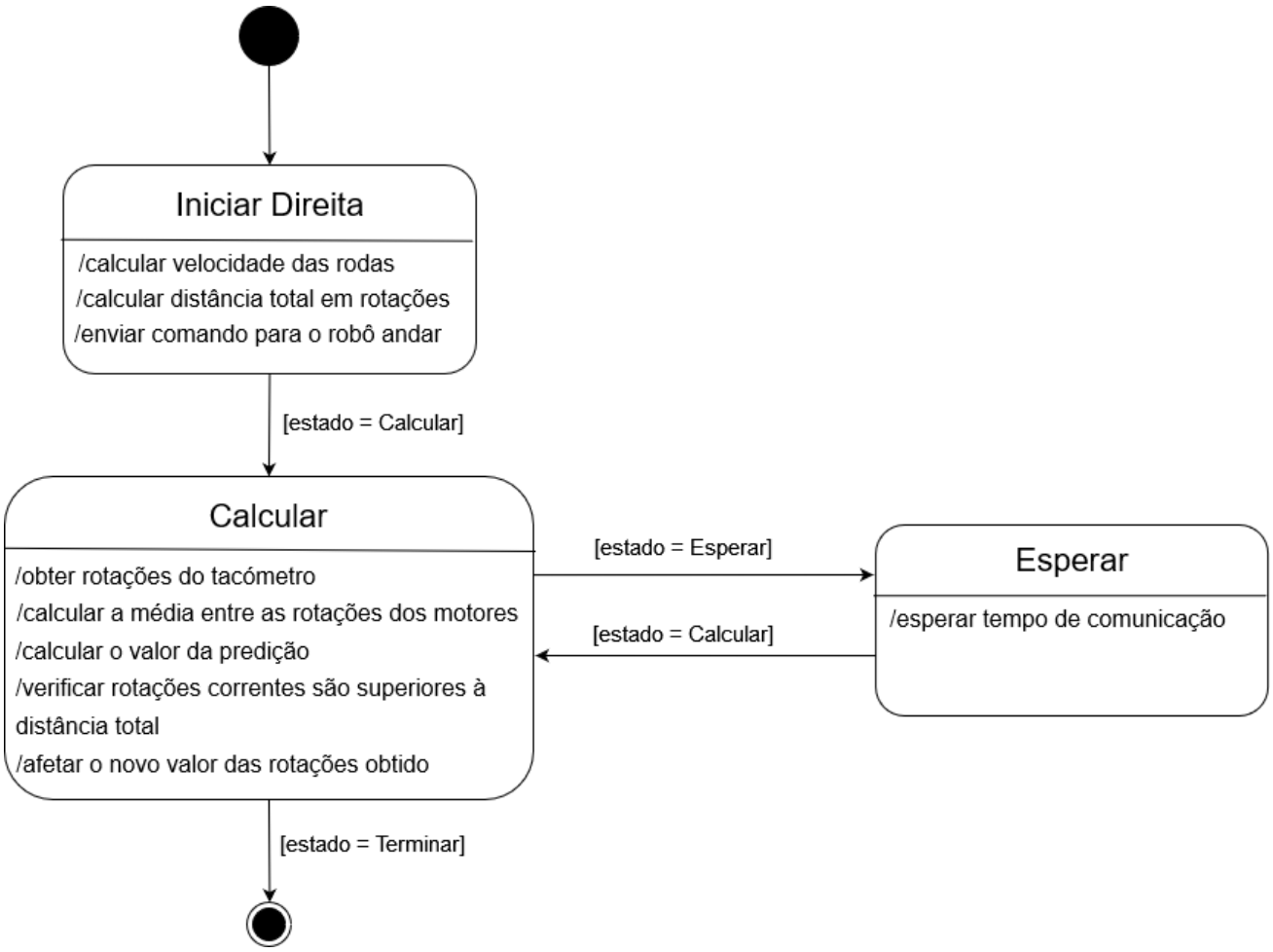

Figura 4 - diagramas de estados do método curve() da curva para a direita

No diagrama de estados apresentado observa-se que o autómato é iniciado no estado Iniciar Direita. Neste caso exemplifica-se quando a curva é para a direita, contudo o curvar para a esquerda é exatamente igual ao curvar direita mudando só as fórmulas dentro do estado Iniciar Direita e o nome do estado seria Iniciar Esquerda.

- Estado Iniciar Direita - neste estado é feito o cálculo da velocidade de cada roda com a fórmula referida anteriormente. Também é calculado a quantidade total de rotações necessárias para completar o curvar, que foi referido como $\hat{A}ngulo_{total}$. Por fim é enviado o comando para o robô curvar com as velocidades calculadas anteriormente.
- Estado Calcular – neste estado primeiramente é obtido as rotações realizadas pelo robô nos dois motores. É necessário de ambos os motores e não somente um, porque eles possuem velocidades diferentes resultando em rotações diferentes. Para verificar se o robô já andou a distância é feito a média das rotações dos dois motores. No cálculo da

verificação da distância pretendida é utilizado uma técnica de predição, para reduzir o erro. O cálculo é feito guardando o número de rotações na iteração anterior e fazendo a diferença entre as rotações corrente menos as rotações anteriores. De seguida é verificado se as rotações obtidas no instante atual são superiores ao $\hat{A}ngulo_{total}$, se sim termina o método mudando o seu estado para Terminar, senão muda para o estado Esperar.

- Estado Esperar – estado responsável para não sobrecarregar o robô com muito pedidos do RotationCount(). O valor de espera escolhido pelo grupo foi 200 milissegundos, que é um valor ligeiramente superior ao tempo mínimo de comunicação.

3.4.4.2 Implementação das contas do curvar à esquerda

$$Distância_{Roda\ direita} = Raio\ círculo + \frac{Distância\ entre\ rodas}{2}$$

$$Distância_{Roda\ esquerda} = Raio\ círculo - \frac{Distância\ entre\ rodas}{2}$$

$$fator = \frac{Distância_{Roda\ direita}}{Distância_{Roda\ esquerda}}$$

$$Velocidade_{roda\ esquerda} = \frac{Velocidade\ robô * 2}{1 + fator}$$

$$Velocidade_{roda\ direita} = Velocidade_{roda\ esquerda} * fator$$

3.3.4.3 Erro associado ao curvar

	Raio	Ângulo	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5
Direita	10	45	C	C	C	C	C
Esquerda	10	45	C	C	+	+	C
Direita	10	90	C	C	+	C	C
Esquerda	10	90	C	-	C	C	C

Direita	10	0	(+/-) 5°	(+/-) 5°	(+/-) 5°	(+/-) 5°	(+/-) 5°
Esquerda	10	0	(+/-) 4°	(+/-) 4°	(+/-) 4°	(+/-) 4°	(+/-) 4°

Tabela 2 - erro associado ao método curve()

Da mesma forma que a reta, também para a curva foi utilizado velocidade 50 e sempre o robô Lego EV3 com o nome EVB. Na tabela anterior o valor C corresponde ao valor correto, o valor + corresponde aquando faz um valor maior que o ângulo pretendido e – para valores inferiores aos pretendidos. Foi utilizado estes símbolos pois o grupo não tinha forma de calcular os ângulos realizados e por isso foi feito a olho. O símbolo +/- é significado que é um valor aproximado, também porque o grupo não conseguia ver os graus realizados.

3.3.5 Implementação do método stop()

Para fazer o robô parar é utilizado o método stop(). Este método possui somente um tipo de paragem que é a paragem após o robô completar o comando que está a executar. Foi optada por somente implementar este tipo de paragem para facilitar a implementação dos outros métodos. Pois seria necessário estar sempre a validar se o estado é terminal ou não complicando a máquina de estados.

O comando utilizado para mandar o robô parar foi o Off(). Foi escolhido este comando em vez do comando Float(), porque durante os testes a primeira vez que era utilizado o comando Float() este possuía um grande erro. Onde percorria uma grande distância mesmo após ter sido enviado o comando para parar. Após ser enviado o comando para parar é utilizado o método ResetAll() para recomeçar o contador de rotações dos motores e conseguir realizar mais comandos.

Por outro lado, a implementação da biblioteca robotLegoEV3.jar possui duas paragens, uma paragem que pára o robô logo após ser enviado o comando e outro que pára o robô após o robô completar o comando que está a realizar.

3.3.6 Exclusão mútua no acesso aos métodos

Todos os métodos implementados na classe MyRobotLego possuem exclusão mútua no acesso ao método. Isto deve-se ao facto de que cada comando é lançado como uma Thread para a interface gráfica não ficar congelada no lançamento de um comando. Desta forma os outros comandos só são executados após o método detentor do semáforo acabar a sua execução. Contudo existe ainda um problema que não ficou resolvido que é a ordem causal dos comandos, porque se existirem duas Threads em espera não é garantido que a Thread a ficar primeiro em espera será a primeira a executar o código. Para resolver este problema teria de se utilizar semáforos específicos para cada método, contudo isso teria uma maior complexidade e não é o foco do projeto estes temas. Esta abordagem também previne erros por exemplo quando o robô está a realizar um comando e se desliga o canal comunicação Bluetooth, desta forma ele só quando acabar o comando é que pára o robô e só depois de parar é que desliga o canal de comunicação.

3. Conclusões

Em suma, com a realização do presente trabalho prático foi possível desenvolver os seguintes conhecimentos:

- Utilização do robô Lego EV3 para desenvolver conhecimentos na área da robótica.
- Utilização de modelos matemáticos, nomeadamente pneumática, para desenvolver formas de pôr o robô a andar em linha reta e em curva.
- Utilização da biblioteca Swing para desenvolver uma interface gráfica para ligar, desligar e enviar comandos para o robô.
- Utilização de máquina de estados para facilitar a elaboração do código funcional dos comandos do robô.
- Utilização de técnicas de predição para redução do erro associado a cada comando.
- Verificação do erro associado aos comandos para ter uma melhor perceção de qual pode vir a ser o comportamento do robô.

4. Bibliografia

[1] Folhas Pais J., Introdução à Robótica e Robot EV3 da Lego, ISEL, 2021/22

5. Anexo

```
import javax.swing.JFrame;
import javax.swing.JButton;

import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

import java.awt.Color;
import java.awt.Font;
import javax.swing.JRadioButton;
import javax.swing.JCheckBox;

public class GuiRobot implements Runnable{

	private JFrame frame;

	private boolean connected = false;

	// text fiels to write distances
	private JTextField fieldRobotName;
	private JTextField fieldDistance;
	private JTextField fieldRay;
	private JTextField fieldAngle;
	private JTextField fieldConsole;


	// buttons to execute commands
	private JButton btnFront;
	private JButton btnRight;
	private JButton btnLeft;
	private JButton btnBack;
	private JButton btnStop;
```

```
private JRadioButton onOff;
private JCheckBox checkBoxDebug;

// variables used
private Variables variables;


public static void main(String[] args) {
	// runs the GUI as thread to not block the commands
	GuiRobot gr = new GuiRobot();
	Thread t = new Thread(gr);
	t.start();
}


public GuiRobot() {
	variables = new Variables();
}

public void run() {
	initializeGui();
}

private void initializeGui() {
	frame = new JFrame();
	frame.setBounds(100, 100, 580, 495);
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frame.getContentPane().setLayout(null);

	// create the elements of the interface
	createButtons();
	createRobotName();
	createDistance();
	createOnOffBtn();
	createRay();
	createAngle();
	createConsole();
```

```
		createDebug();


		frame.setVisible(true);
	}


	// create all the buttons
	private void createButtons() {
		// left button
		btnLeft = new JButton("Esquerda");
		btnLeft.setBounds(79, 207, 140, 68);
		btnLeft.setBackground(new Color(19, 56, 190));
		btnLeft.setEnabled(false);
		frame.getContentPane().add(btnLeft);

		// right button
		btnRight = new JButton("Direita");
		btnRight.setBounds(361, 207, 140, 68);
		btnRight.setBackground(new Color(230, 219, 172));
		btnRight.setEnabled(false);
		frame.getContentPane().add(btnRight);

		// stop button
		btnStop = new JButton("Parar");
		btnStop.setBounds(220, 207, 140, 68);
		btnStop.setBackground(new Color(227, 36, 43));
		btnStop.setEnabled(false);
		frame.getContentPane().add(btnStop);

		// front button
		btnFront = new JButton("Frente");
		btnFront.setBounds(220, 138, 140, 68);
		btnFront.setBackground(new Color(61, 237, 151));
		btnFront.setEnabled(false);
		frame.getContentPane().add(btnFront);

		// back button
```

```
        btnBack = new JButton("Retaguarda");
        btnBack.setBounds(220, 276, 140, 68);
        btnBack.setBackground(new Color(182, 95, 207));
        btnBack.setEnabled(false);
        frame.getContentPane().add(btnBack);

        // listeners for the buttons
        listenerBtnLeft();
        listenerBtnRight();
        listenerBtnFront();
        listenerBtnBack();
        listenerBtnStop();

        // listener when close the connection with robot
        listenerOnClose();
    }

    private void createOnOffBtn() {
        // Radio button to turn robot on or off
        onOff = new JRadioButton("On/Off");
        onOff.setFont(new Font("Tahoma", Font.PLAIN, 15));
        onOff.setBounds(410, 25, 109, 23);
        onOff.setSelected(variables.isOnOff());
        frame.getContentPane().add(onOff);

        listenerConnectRobot();
    }

    // create the text and field to write the name
    private void createRobotName() {
        // label to show text of the robot name
        JLabel labelRobot = new JLabel("Robot:");
        labelRobot.setFont(new Font("Tahoma", Font.PLAIN, 19));
        labelRobot.setBounds(183, 13, 67, 37);
        frame.getContentPane().add(labelRobot);

        // field to set the robot name
        fieldRobotName = new JTextField("" + variables.getNomeRobot());
```

```
		fieldRobotName.setBounds(245, 21, 90, 25);
		frame.getContentPane().add(fieldRobotName);

		listenerFieldRobotName();
	}

	// text and field of distance
	private void createDistance() {
		// label to show text of distância
		JLabel labelDistance = new JLabel("Dist\u00E2ncia:");
		labelDistance.setFont(new Font("Tahoma", Font.PLAIN, 16));
		labelDistance.setBounds(10, 92, 80, 23);
		frame.getContentPane().add(labelDistance);

		// field to write the distance
		fieldDistance = new JTextField("" + variables.getDistance());
		fieldDistance.setBounds(85, 92, 86, 25);
		frame.getContentPane().add(fieldDistance);

		listenerDistance();
	}

	private void createRay() {
		// text to show raio
		JLabel labelRay = new JLabel("Raio:");
		labelRay.setFont(new Font("Tahoma", Font.PLAIN, 16));
		labelRay.setBounds(211, 95, 67, 17);
		frame.getContentPane().add(labelRay);

		// field to write the ray
		fieldRay = new JTextField("" + variables.getRadius());
		fieldRay.setBounds(250, 92, 86, 25);
		frame.getContentPane().add(fieldRay);

		listenerRay();
	}

	private void createAngle() {
```

```
		// text to write ângulo
		JLabel labelAngle = new JLabel("\u00C2ngulo:");
		labelAngle.setFont(new Font("Tahoma", Font.PLAIN, 16));
		labelAngle.setBounds(403, 95, 58, 20);
		frame.getContentPane().add(labelAngle);

		// field to write the angle
		fieldAngle = new JTextField("" + variables.getAngle());
		fieldAngle.setBounds(465, 92, 86, 25);
		frame.getContentPane().add(fieldAngle);

		listenerAngle();
	}

	private void createConsole() {
		JLabel labelConsole = new JLabel("Consola");
		labelConsole.setFont(new Font("Tahoma", Font.PLAIN, 15));
		labelConsole.setBounds(72, 370, 58, 20);
		frame.getContentPane().add(labelConsole);

		fieldConsole = new JTextField();
		fieldConsole.setBounds(72, 390, 458, 40);
		fieldConsole.setEditable(false);
		frame.getContentPane().add(fieldConsole);
	}

	private void createDebug() {
		checkBoxDebug = new JCheckBox("Debug");
		checkBoxDebug.setBounds(72, 334, 97, 23);
		checkBoxDebug.setSelected(variables.isDebug());
		frame.getContentPane().add(checkBoxDebug);

		listenerToDebug();
	}

	private void listenerFieldRobotName() {
		fieldRobotName.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
```

```
                        variables.setNomeRobot(fieldRobotName.getText());
                        myPrint("Nome:"+ variables.getNomeRobot());
                  }
            });
      }

      private void listenerBtnLeft() {
            btnLeft.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        new Thread() {
                              public void run() {
                                    myPrint("Esquerda: " +
String.valueOf(variables.getAngle()) + " "+
String.valueOf(variables.getRadius()));
                                    // TODO
                                    if (variables.getMyImplentation()) {

      variables.getMyRobot().curveLeft(variables.getAngle(),
variables.getRadius());


                                    } else {

      variables.getRobot().CurvarEsquerda(variables.getRadius(),
variables.getAngle());
                                          variables.getRobot().Parar(false);
                                    }
                              }
                        }.start();
                  }
            });
      }

      private void listenerBtnRight() {
            btnRight.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        new Thread() {
                              public void run() {
                                    myPrint("Direita: " +
String.valueOf(variables.getAngle()) + " "+
String.valueOf(variables.getRadius()));
                                    // TODO
```

```
                                    if (variables.getMyImplentation()) {

      variables.getMyRobot().curveRight(variables.getAngle(),
variables.getRadius());
                                    } else {

      variables.getRobot().CurvarDireita(variables.getRadius(),
variables.getAngle()); // TODO
                                          variables.getRobot().Parar(false);
                                    }
                              }
                        }.start();
                  }
            });
      }

      private void listenerBtnFront() {
            btnFront.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        // creates a thread that makes the robot goes forward
                        new Thread() {
                            public void run() {
                              myPrint("Frente: " + variables.getDistance());
                                    // TODO
                                    if (variables.getMyImplentation()) {

      variables.getMyRobot().line(variables.getDistance());

                                    } else {

      variables.getRobot().Reta(variables.getDistance());
                                          variables.getRobot().Parar(false);
                                    }
                              }
                        }.start();
                  }
            });
      }

      private void listenerBtnStop() {
```

```
		btnStop.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				new Thread() {
					public void run() {
						myPrint("Parar");
						// TODO
						if (variables.getMyImplentation())
							variables.getMyRobot().stop();

						else
							variables.getRobot().Parar(true);
					}
				}.start();
			}
		});
	}

	private void listenerBtnBack() {
		btnBack.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				new Thread() {
					public void run() {
						myPrint("Retaguarda: " +
String.valueOf(variables.getDistance()));
						// TODO
						if (variables.getMyImplentation()) {

	variables.getMyRobot().line(variables.getDistance() * (-1));
						} else {

	variables.getRobot().Reta(variables.getDistance() * (-1));
							variables.getRobot().Parar(false);
						}
					}
				}.start();
			}

		});
	}
```

```
	private void listenerConnectRobot() {
		onOff.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if (onOff.isSelected()) {
					myPrint("Conexão com robô");
					//TODO
					if(variables.getMyImplentation()) connected
=variables.getMyRobot().openEV3(variables.getNomeRobot());

					else connected =
variables.getRobot().OpenEV3(variables.getNomeRobot());

					// set the interface state
					setButtinsState(connected);
				}
				else {
					myPrint("Disconexão com robô");
					// TODO
					if(variables.getMyImplentation())
variables.getMyRobot().closeEV3();

					else variables.getRobot().CloseEV3();

					onOff.setSelected(false);
					variables.setOnOff(false);
					connected = false;
					// set the interface state
					setButtinsState(false);
				}
				try {
					Thread.sleep(100);
				} catch (InterruptedException e1) {
					e1.printStackTrace();
				}

			}
		});
	}
```

```
	public void listenerOnClose() {
		frame.addWindowListener(new WindowAdapter() {
		    @Override
		    public void windowClosing(WindowEvent windowEvent) {
		      if(connected) {
				// disconnect the robot
				if(variables.getMyImplentation())
variables.getMyRobot().closeEV3();

				else variables.getRobot().CloseEV3();
		      }
		    }
		});
	}

	private void listenerDistance() {
		fieldDistance.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				// verifies if the sting only contains numbers, otherwise
shows a message
				if(fieldDistance.getText().matches("[0-9]*")) {
					int dist =
Integer.parseInt(fieldDistance.getText());
					if(dist <= 400) {
						variables.setDistance(dist);
						myPrint("Distancia:"+
variables.getDistance());
					}
					else JOptionPane.showMessageDialog(frame, "A
distância só pode ser no máximo 400");//
				}

				else JOptionPane.showMessageDialog(frame, "A distância só
pode conter números");//

			}
		});
	}
```

```
	private void listenerRay() {
		fieldRay.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(fieldRay.getText().matches("[0-9]*")) {
					int radius = Integer.parseInt(fieldRay.getText());
					if(radius <= 50) {
						variables.setRadius(radius);
						myPrint("Raio:"+ variables.getRadius());
					}
					else JOptionPane.showMessageDialog(frame, "O raio
só pode ser no máximo 50");//
				}

				else JOptionPane.showMessageDialog(frame, "O raio só pode
conter números");//
			}
		});
	}

	private void listenerAngle() {
		fieldAngle.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(fieldAngle.getText().matches("[0-9]*")) {
					int angle = Integer.parseInt(fieldAngle.getText());
					if(angle <= 720) {
						variables.setAngulo(angle);
						myPrint("Angulo:"+ variables.getAngle());
					}
					else JOptionPane.showMessageDialog(frame, "O angulo
só pode ser no máximo 720");//
				}
				else JOptionPane.showMessageDialog(frame, "O angulo só
pode conter números");//
			}
		});
	}

	private void listenerToDebug() {
		checkBoxDebug.addActionListener(new ActionListener() {
```

```
                public void actionPerformed(ActionEvent e) {
                      variables.setDebug(checkBoxDebug.isSelected());
                }
          });
    }


    private void myPrint(String str) {
          if (variables.isDebug())
                fieldConsole.setText(str);
    }

    public void setButtinsState(boolean result) {
          onOff.setSelected(result);
          variables.setOnOff(result);
          btnLeft.setEnabled(result);
          btnRight.setEnabled(result);
          btnFront.setEnabled(result);
          btnStop.setEnabled(result);
          btnBack.setEnabled(result);
    }
}
```

```
public class Variables {

	// variable used to define if will use the prof library or
	// our implementation
	private final boolean MY_IMPLEMENTATION = true;

	private String robotName;
	private boolean onOff, debug;
	private int radius, angle, distance;
	private RobotLegoEV3 robot;
	private MyRobotLego myRobot;

	public Variables() {
		robotName= "EVB";
		onOff = false;
		debug = true;
		radius = 10;
		angle = 90;
		distance = 20;
		robot = new RobotLegoEV3();
		myRobot = new MyRobotLego();
	}

	public String getNomeRobot() {
		return robotName;
	}

	public void setNomeRobot(String nomeRobot) {
		this.robotName = nomeRobot;
	}

	public boolean isOnOff() {
		return onOff;
	}

	public void setOnOff(boolean onOff) {
		this.onOff = onOff;
	}

	public boolean isDebug() {
		return debug;
	}

	public void setDebug(boolean debug) {
		this.debug = debug;
	}

	public int getRadius() {
		return radius;
	}

	public void setRadius(int radius) {
		this.radius = radius;
	}

	public int getAngle() {
```

```
		return angle;
	}

	public void setAngulo(int angle) {
		this.angle = angle;
	}

	public int getDistance() {
		return distance;
	}

	public void setDistance(int distance) {
		this.distance = distance;
	}

	public RobotLegoEV3 getRobot() {
		return this.robot;
	}

	public MyRobotLego getMyRobot() {
		return this.myRobot;
	}

	public boolean getMyImplentation() {
		return this.MY_IMPLEMENTATION;
	}

}
```

```
import java.util.Arrays;
import java.util.concurrent.Semaphore;

public class MyRobotLego {

	private final int VELOCITY = 50;

	private InterpretadorEV3 interpretador;

	private final float WHEEL_RADIUS = 2.73f;
	// total distance between wheels
	private final float WHEEL_CENTER_MASS = 9.5f;

	// multiple possible states, to the line and curve
	private enum states {START, LEFT, RIGHT, CALCULATE, WAIT, END};

	// time of bluetooth to make communication
	private final int COMMUNICATION_TIME = 200;

	// obtain the right wheel
	private final int RIGHT_WHEEL = InterpretadorEV3.OUT_C;
	private final int LEFT_WHEEL = InterpretadorEV3.OUT_B;
	private final int RIGHT_LEFT_WHEELS = InterpretadorEV3.OUT_BC;

	// to prevent multiple threads access the multiple commands simultaneously
	private Semaphore semaphore;

	public MyRobotLego() {
		interpretador = new InterpretadorEV3();
		semaphore = new Semaphore(1);
	}

	// open communication with robot EV3
	public boolean openEV3(String robotName) {
		// because its used multi-threading, only one access at the same
time
		try {
			semaphore.acquire();
		} catch (InterruptedException e1) {
```

```
                e1.printStackTrace();
        }
        // connects to the robot
        boolean result = this.interpretador.OpenEV3(robotName);
        // releases the other threads
        semaphore.release();
        return result;
    }

    // close the communication with robot EV3
    public void closeEV3() {
        // because its used multi-threading
        try {
                semaphore.acquire();
        } catch (InterruptedException e1) {
                e1.printStackTrace();
        }

        // remove all the commands previously sent
        this.interpretador.ResetAll();
        try {
                Thread.sleep(COMMUNICATION_TIME);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        this.interpretador.CloseEV3();
        // releases the thread to another thread uses
        semaphore.release();
    }

    // realizes the straight line
    // positive values go forward, negative values goes backwards
    public void line(int distance) {
        // this way a thread will only start after do a command
        try {
                semaphore.acquire();
        } catch (InterruptedException e1) {
                e1.printStackTrace();
```

```
		}

		// angle that must be done by the robot
		double ang = 0;

		states currentState = states.START;
		// when the distance is positive goes forward
		// if negative goes backwards
		int signal = (distance > 0) ? 1 : -1;

		// variable used to prediction technique
		int deltaActual = 0;

		while (currentState != states.END) {
			switch (currentState) {
			case START:
				// the value of the distance must be positive
				// so we use the abs to set positive the value
				ang = (Math.abs(distance) / WHEEL_RADIUS) * 180 /
Math.PI;
				System.out.println("Angle in degrees: " + ang);

				// sends the message to go forward
				this.interpretador.OnFwd(RIGHT_WHEEL, VELOCITY * signal,
LEFT_WHEEL, VELOCITY * signal);
				currentState = states.CALCULATE;
				break;

			case CALCULATE:
				int rotationCounts =
this.interpretador.RotationCount(RIGHT_WHEEL);

				// because the distance may be negative and the angles
must always be positive
				rotationCounts = rotationCounts * signal;
				System.out.println("Rot: " + rotationCounts);

				int deltaD = rotationCounts - deltaActual;
				System.out.println("DeltaD: " + deltaD);
```

```
                        // if the state is end to stops
                        // or the rotation count completes the task
                        //currentState = (rotationCounts + deltaD > ang ) ?
states.END : states.WAIT;
                        currentState = (rotationCounts > ang ) ? states.END :
states.WAIT;
                        // sets the delta to the current rotations, to predict
the value and gets the
                        // least error possible
                        deltaActual = rotationCounts;
                        break;

                  case WAIT:
                        // to not overload the robot with messages
                        // must have a delay
                        try {
                              Thread.sleep(COMMUNICATION_TIME);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        currentState = states.CALCULATE;
                        break;

                  }
            }
            // after complete of the command sends a release to other thread
starts running
            semaphore.release();

            //if no one sends the command to stop it keep on going until receive
a command to stop
            //stop();
      }

      // angle is in degrees, and radius is in cm
      public void curveRight(int angle, int radius) {
            // this way a thread will only start after do a command
            try {
                  semaphore.acquire();
```

```
            } catch (InterruptedException e) {
                  e.printStackTrace();
            }

            // makes the right curve
            curve(true, angle, radius);



            // after complete the command gives one permission to another thread
is freed
            semaphore.release();

            // if no one sends a stop command it keeps going until send another
command or
            // stop command
            //stop();
      }


      public void curveLeft(int angle, int radius) {
            // this way a thread will only start after do a command
            try {
                  semaphore.acquire();
            } catch (InterruptedException e) {
                  e.printStackTrace();
            }

            // makes the left curve
            curve(false, angle, radius);

            // after complete the command gives one permission to another thread
is freed
            semaphore.release();
            // if no one sends a stop command it keeps going until send another
command or stop command
            //stop();
      }

      // creates a curve with state of curve right and left
```

```
	private void curve(boolean right, int angle, int radius) {

		states currentState = (right) ? states.RIGHT : states.LEFT;

		double ang = 0;
		// variable used to prediction technique
		double deltaActual = 0;

		while (currentState != states.END) {
			switch (currentState) {
			case RIGHT:
				// calculations used to curve right
				double radiusRight = radius - (WHEEL_CENTER_MASS / 2);
				double radiusLeft = radius + (WHEEL_CENTER_MASS / 2);
				double factor = radiusLeft / radiusRight;

				int rightSpeed = (int) Math.round((VELOCITY * 2) / (1 +
factor));
				int leftSpeed = (int) Math.round((factor * rightSpeed));
				System.out.println("left: " + leftSpeed + ", right: " +
rightSpeed);

				double totalDist = angle * radius;

				ang = (Math.abs(totalDist) / WHEEL_RADIUS);
				System.out.println("TOTAl angle: " + ang);

				// the right wheel is C, and left
				this.interpretador.OnFwd(RIGHT_WHEEL, rightSpeed,
LEFT_WHEEL, leftSpeed);
				currentState = states.CALCULATE;
				break;

			case LEFT:
				// calculations used to curve left
				radiusRight = radius + (WHEEL_CENTER_MASS / 2);
				radiusLeft = radius - (WHEEL_CENTER_MASS / 2);
				factor = radiusRight / radiusLeft;
				leftSpeed = (int) Math.round((VELOCITY * 2) / (1 +
```

```
factor));
                        rightSpeed = (int) Math.round((factor * leftSpeed));
                        System.out.println("left: " + leftSpeed + ", right: " +
rightSpeed);



                        totalDist = angle * radius;

                        ang = (Math.abs(totalDist) / WHEEL_RADIUS);
                        System.out.println("TOTAl ang: " + ang);

                        // the right wheel is C, and left
                        this.interpretador.OnFwd(RIGHT_WHEEL, rightSpeed,
LEFT_WHEEL, leftSpeed);
                        currentState = states.CALCULATE;
                        break;

                case CALCULATE:
                        int[] rotationCounts =
this.interpretador.RotationCount(RIGHT_WHEEL, LEFT_WHEEL);
                        double result = Math.round((rotationCounts[0] +
rotationCounts[1]) / 2);

                        System.out.println("Rots: " +
Arrays.toString(rotationCounts));
                        System.out.println("Res: " + result);

                        double prediction = result - deltaActual;

                        // verify if the robot already complete the curve or not
with prediction
                        currentState = (result + prediction > ang) ? states.END :
states.WAIT;

                        // used to store the previous value and make movement
prediction
                        deltaActual = result;
                        break;

                case WAIT:
                        // to not overload the robot with messages
```

```
                        // must have a delay
                        try {
                              Thread.sleep(COMMUNICATION_TIME);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        currentState = states.CALCULATE;
                        break;
                  }
            }
      }

      // stops the robot
      // it can also uses the Float command, but it gets a bigger error
      // and the first time it as used gets a really high error
      public synchronized void stop() {
            try {
                  semaphore.acquire();
                  //this.interpretador.Float(RIGHT_LEFT_WHEELS);
                  this.interpretador.Off(RIGHT_LEFT_WHEELS);
                  Thread.sleep(COMMUNICATION_TIME);
                  this.interpretador.ResetAll();

            } catch (InterruptedException e) {
                  e.printStackTrace();
            }
            semaphore.release();
      }
}
```