Inteligência Artificial – 3º Projeto

Fuzzy Systems/Árvores de Decisão/Redes Neuronais | 2º Semestre 2021/22

A resolução deste projeto deve ser feita em grupos de dois alunos e tem de ser realizada em Java. A entrega do mesmo deve ser a partir da extração direta do eclipse (ou IDE equivalente) para uma pasta comprimida cujo nome deve ser os números de aluno seguido dos respetivos primeiro e último nome (exemplo: "12345_JoaoSilva_54321_PedroAlmeida.zip").

A entrega é realizada através do e-learning até **23:59** de **22/05/2022**.

Controlador de robot detetor de cogumelos

Pretende-se implementar um sistema que permita controlar um robot detetor de cogumelos comestíveis num campo agrícola, evitando a recolha de cogumelos venenosos.

Desta forma o robot vai receber informação sobre o cogumelo detetado e decidirá através de um algoritmo supervisionado de árvores de decisão ou de redes neuronais se colhe ou não esse cogumelo. O robot recebe a informação através de 3 sensores: um central, um do lado esquerdo e um do lado direito que permitem a deteção dos cogumelos. Os sensores têm um alcance de 10 m e estão distribuídos como demonstrado na Figura 1:

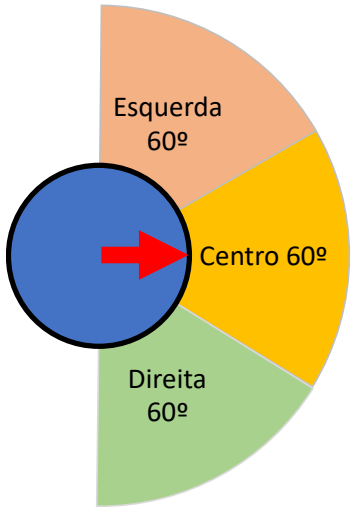

Figura 1 - Robot e os seus sensores de obstáculos

Para **cada um dos sensores,** devem ser definidos 3 conjuntos vagos (*fuzzy sets*): *near*, *medium* e *far.* O universo destas variáveis vai de 0 a 10. Adicionalmente, deverão ser criados conjuntos vagos que representem a velocidade do robot (*slow, medium, fast*) e a classificação de um cogumelo (*poisonous, none, edible*).

O sistema deve calcular o ângulo de rotação das rodas do robot e a ação a ser realizada (colher cogumelo, destruir ou não fazer nada). O universo da variável que calcula o **ângulo das rodas** vai de -45º a 45º e devem ser definidos 5 conjuntos vagos para a mesma: *strong-left, left, center, right e strong-right*. Para a variável da **ação a ser realizada** deverão ser definidos 3 conjuntos vagos: *no_action, destroy e pick-up*.

a) Usando API jFuzzyLogic, crie um ficheiro do tipo *.fcl* que represente o sistema anteriormente mencionado e inclua as regras necessárias para controlar o robot.
b) Crie as classes java necessárias para a implementação deste sistema. Deverão ser construídos classificadores baseados em árvores de decisão (J48) ou redes neuronais (MLP) (um dos dois).

Os classificadores deverão ser treinados com base no dataset fornecido: **mushrooms.arff.** Este dataset apresenta três características sobre cogumelos: o seu odor, a cor da sua esporada (spore-print-color), a sua forma (cap-shape) e a sua classe (comestível-*edible* ou venenoso-*poisonous*).

De seguida, estão representados os valores para cada atributo:

Atributo	Valores
Class (Target)	{poisonous,edible}
Odor	{almond,anise,creosote,fishy,foul,musty,none,pungent,spicy}
Spore-print-color	{black,brown,buff,chocolate,green,orange,purple,white,yellow}
Cap-Shape	{bell,conical,convex,flat, knobbed,sunken}

Após o treino do classificador, este deverá classificar as novas instâncias como sendo comestíveis ou venenosas. De acordo com esse resultado, o sistema fuzzy deverá controlar o robot e decidir se o cogumelo deve ser apanhado ou destruído. Idealmente, caso o cogumelo seja venenoso, o robot deverá destruí-lo (*destroy*). Caso seja, comestível, deverá apanhá-lo (*pick-up*). Se não tiver classificação, o robô não fará nada (*no_action*).

Será disponibilizada uma classe java que permite fazer a simulação do robot – Simulator.java. Serão igualmente disponibilizadas as classes Mushroom.java, NewInstances.java e Visualizer.java, assim como o enumerador das ações a serem executadas Action.java.

O GUI do simulador apresenta o número de pontos atuais, assim como a ação que está a ser executada pelo robot naquele momento, como representado na Figura 2.
O esquema de pontuações foi definido como sendo o seguinte: Se o robot apanhar um cogumelo venenoso, perde 10 pontos. Se o destruir, ganha 1 ponto. Se apanhar um cogumelo comestível, ganha 1 ponto. Se destruir um cogumelo comestível perde 5 pontos.

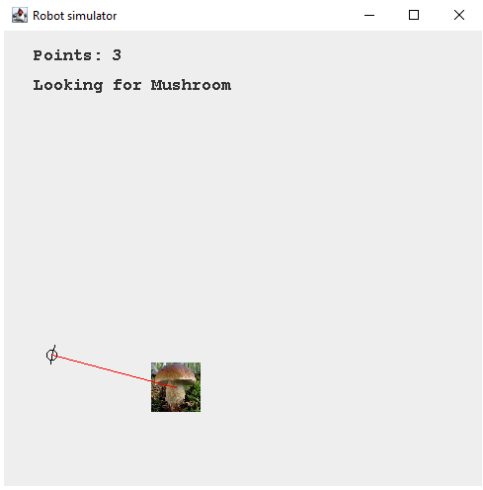

Figura 2 – Janela do simulador com a representação de um cogumelo comestível

Relativamente à classe do simulador, esta disponibiliza os seguintes métodos:

Package simulator

Class Simulator

java.lang.Object

simulator.Simulator

```
public class Simulator
extends java.lang.Object
```

Version: 5

- ***Method Summary***

Modifier and Type	Method	Description
double	getDistanceC()	Getter for the distance of the closest obstacle detected by the central sensor
double	getDistanceL()	Getter for the distance of the closest obstacle detected by the left sensor
double	getDistanceR()	Getter for the distance of the closest obstacle detected by the right sensor
java.lang.String[]	getMushroomAttributes()	Getter for the attributes of the observed mushroom.
int	getRobotSpeed()	Getter for the robot speed.
int	getSimulationSpeed()	Getter for the simulation speed
void	setAction (simulator.Action action)	Setter for the action for the robot to perform in the next time step.
void	setRobotAngle (double angle)	Sets the new angle for the wheels of the robot.
void	setRobotSpeed (int robotSpeed)	Sets the new robot speed.

Modifier and Type	Method	Description
void	setSimulationSpeed (int simulationSpeed)	Setter for the time needed for a simulation step.
void	step()	Moves simulation one time step forward and shows the new state of the environment in GUI.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

- ***Constructor Details***

  - **Simulator**

    public Simulator()

- ***Method Details***

  - **setRobotAngle**

    public void setRobotAngle(double angle)

    Sets the new angle for the wheels of the robot.

    Parameters:

    angle - the value of the new angle in degrees

  - **setAction**

    public void setAction(simulator.Action action)

    Setter for the action for the robot to perform in the next time step. Possible actions are:

    - DESTROY: used to instruct the robot to destroy the mushroom,

    - NO_ACTION: the robot does not perform any actions,

    - PICK_UP: used to instruct the robot to pick up the mushroom.

    Parameters:

`action` - to be performed.

- **getRobotSpeed**

```
public int getRobotSpeed()
```

Getter for the robot speed.

Returns: the current speed of the robot

- **getMushroomAttributes**

```
public java.lang.String[] getMushroomAttributes()
```

Getter for the attributes of the observed mushroom.

Returns:

an array with the mushroom's attributes

- **setRobotSpeed**

```
public void setRobotSpeed(int robotSpeed)
```

Sets the new robot speed.

Parameters:

`robotSpeed` - the new value for the robot speed.

- **getSimulationSpeed**

```
public int getSimulationSpeed()
```

Getter for the simulation speed

Returns:

time for a simulation step in milliseconds.

- **setSimulationSpeed**

```
public void setSimulationSpeed(int simulationSpeed)
```

Setter for the time needed for a simulation step.

Parameters:

`simulationSpeed` - time in milliseconds.

- **step**

```
public void step()
```

Moves simulation one time step forward and shows the new state of the environment in GUI. Uses the current angle of the wheels to move the robot. After the new position of the robot has been computes the sensor values are updated.

- **getDistanceC**

```
public double getDistanceC()
```

Getter for the distance of the closest obstacle detected by the central sensor

Returns:

distance in meters of the closest obstacle inside the opening angle (60º) of the central sensor.

- **getDistanceL**

```
public double getDistanceL()
```

Getter for the distance of the closest obstacle detected by the left sensor

Returns:

distance in meters of the closest obstacle inside the opening angle (60º) of the left sensor.

- **getDistanceR**

```
public double getDistanceR()
```

Getter for the distance of the closest obstacle detected by the right sensor

Returns:

distance in meters of the closest obstacle inside the opening angle (60º) of the right sensor.