UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Simulation Biased Random Walk associated with Escherichia Coli

Final Degree Project

Computer Science

Author: Jose Miguel Lloret Pérez

Supervisors: Dr. Ángel Valera Fernandez

Dr. John McCall

August 2, 2012

In collaboration with:

ROBERT GORDON UNIVERSITY•ABERDEEN

Abstract

The aim of the project is to simulate cellular intelligence in e-puck robots through a novel method named Artificial Reaction Network (ARN). Biased Random Walk (BRW) is implemented in three robots simultaneously and these are controlled by a computer which runs a program in Java platform. The characteristics and limitations that e-puck has and protocol for communicating Java platform with e-puck is discussed. Also approaches like: Odometry, Kalman Filter (KF), Particle Filter (PF), proximal and distal are presented as well as its advantages and disadvantages. Thus, the implementation of each approach selected is explained and future improvements are discussed.

Acknowledgements

The author wishes to express his gratitude to his supervisor, John McCall who has offered invaluable assistance, support and guidance. Also, the author would like to thank Claire Gerrard since without her knowledge, this study would not have been successful. Not forgetting David Davidson for giving him the opportunity of be an exchange student in this University as well as the students met at Ideas Research Group. The author whishes to express his love to Cinta Gomez for her understandability in every situation elapsed during the project since without her would not have been possible to get the strenght enough for achieving the purposes. Finally, the author is grateful for his family who has been giving him help and backing in his experience far away from home.

Contents

List of Figures

List of Tables

Listings

Introduction

Artificial Intelligence (AI) is the study of intelligence machines and is one of the branches of computing which was invented by John McCarthy and defined as he said "the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable" [7]. Since then, many approaches to simulate intelligence have emerged, such as bio-inspired approaches which have achieved that AI has enormous progress.

Bio-inspired AI approaches have proven valuable, for example: Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Ant Colony Optomization (ACO).

This project intends to explore the potential of Artificial Reaction Network (ARN) as a source to simulate cellular intelligence in a particular robot. The Ideas Research Institute is part of Robert Gordon University and could consider this type of project like the first project using e-puck robot, an educational robot with an emerging use in Engineering.

The first aim of this project has been to establish a contact with every term involved with the robot, studying its characteristics, as well as its limitations.

The secondary aim has been to research how to communicate the robot with programs in java, so this may be very useful for a future research projects related to simulated cellular intelligence using Java as a principal programming language, due to the extensibility and power that resides in this language.

The third aim has been to apply ARN to simulate the Biased Random Walk (BRW) associated with Escherichia Coli, combining the first aim as well as the secondary aim, to simulate artificial intelligence in e-puck robots.

Requirements analysis

ARN is a novel method of simulating cellular intelligence that can be described as a modular S-System, with some properties in common with other Systems Biology and AI techniques [6]. Its biological basis has been validated using real biochemical data by creating a working model of the Cell Signalling Networks (CSN) of the well characterized signalling network of Escherichia Coli.

ARN is an abstract model of a CSN and are described as networks of biochemical reactions with allow communication, response and feedback within and between the cells. The characteristics of this cellular intelligence like recognition, classification, response, communication, learning and self-organization [5] are the result of these complex networks and are examined as a source of inspiration for development of novel AI techniques using this novel method, ARN.

Escherichia Coli, to highlight the explanation, may be described as unicellular bacteria that present an ideal pathway to explore emergent properties of cell intelligence. This bacterium has chemoreceptors which recognise and bind to specific chemicals within the environment.

BRW is one of the behaviours that Escherichia Coli can be experiment, and may be described as a series of "runs" and "tumbles". "Run" refers to motions in a straight line for a period of time, whilst "Tumble" can be described as a random turn. For the project developed all of these theories are applied like:

- Bacteria bias their walks based on the concentration gradient of a particular chemical and bacteria can be in a good or bad condition depending the position which is it.
- There is an environment that each cell can move in, and there are three cells moving around this environment, where each cell may be in a high concentration or in a low concentration of a determinate characteristic, for instance food.
- The centre of the environment represents the best concentration and when each bacterium moves outwards, this value of concentration decreases till it reaches the bad conditions which correspond to lowest concentration.
- The cell will increase its run length as it moves towards the centre, decreases its run if it moves outwards from the centre, trying to gravitate towards more favourable conditions.

Thus, ARN has been applied to simulate the BRW of Escherichia Coli and indeed it is the reason of these clarifications.

In the following section, "Detailed specification" how to adapt BRW to produce cellular intelligence with a particular robot will be explained.

Detailed specification

BRW is the behaviour chosen to simulate in a real environment using robots. To adapt this cell behaviour, details of the needed requirements will be explained below:

Basic actions

- The robot always starts in a particular point given by x coordinate as well as y coordinate. So, it can be affirmed that the robot is always in a coordinate (x, y) located in a particular environment.
- "Run" is translated as a motion in a straight line where the duration of this depends on the distance between current coordinates and coordinates to reach.
- "Tumble" refers to angle needed to turn the robot from the current coordinates to reach the new coordinates.

Why are the basic actions implemented not according to BRW? Because another program executes these actions and indeed this program is responsible for implementing BRW using ARN as a source. So, the program developed for this project has received a file with different orders to go for each robot, thus this orders are in accordance with the results produced by the program which really implements BRW.

Protocol avoidance

- The number of robots sharing the environment is reduced to three and each one is simulating a cell. Each robot must have implemented the avoiding protocol.
- They need to be able avoid collision with the walls.
- They need avoid crashing each other, taking into account that they probably are in motion.

Concurrence

- There is a computer which controls each robot, hence instances need to be run simultaneously.
- There is need to manage every request which is coming from the computer program, hence various input channel are needed.
- There is need to send orders to robots simultaneously, therefore various output channel are needed.

- The program must be run in Java as the language to exchange information between robots.

Environment to simulate

- The robots must be run in an enclosed surface.
- The width is measured in centimetres and corresponds to x. Also the height is measured in centimetres and corresponds to y.
- There are four walls around the surface, and the height is four centimetres so the robots are expected to avoid crashing with them.
- The ground of the enclosed surface is like a green carpet which is appropriate for the wheels as it is not slippery.

E-puck robot, selected to simulate

The e-puck robot is a differential drive robot designed by Dr. Francesco Mondada and Michael Bonani in 2006 at École Polytechnique Fédérale de Lausanne (EPFL). It was intended for educational use but also has been used for research [20]. The project is based on an open hardware concept; similarly the software developed is fully open source, offering unlimited extension possibilities.

The design of the robot is based on two criteria [8]:

- Desktop size: A robot which can evolve on the desk near to computer improves drastically the student efficiency during experimentation.
- Flexibility: Various fields of education (signal processing, automatic control, embedded programming and distributed intelligent systems, among others) as well as a wide set of functionalities can be applied to this robot.

Overview

The robot e-puck shown in (Figure 1) has a diameter of 75 mm. The user can connect physical extensions to provide additional sensors, actuators or even computational power. There are three physically different types of extensions (top, bottom and sandwich). The height of the e-puck depends on the connected extensions. The battery can be extracted easily from the bottom of the e-puck. The hardware and the software which has this robot as well as mechanisms and useful theories will be explained below.

Hardware

E-puck hardware has an electronic structure (Figure 2) built around a microchip dsPIC microcontroller. This microcontroller has a CPU which runs at 64 MHz and provides 16 Millions Instructions Per Second (MIPS). This processor is supported by a custom tailored version of the GCC C compiler. The random access memory is 8 KB and also has 144 KB of flash memory to store user programs. The e-puck contains various sensors, identified as blue (Figure 2) to ensure a broad range of experimentation possibilities. Details of each one is given below:

- Low battery detection: It is enabled when robot detects low charge battery.

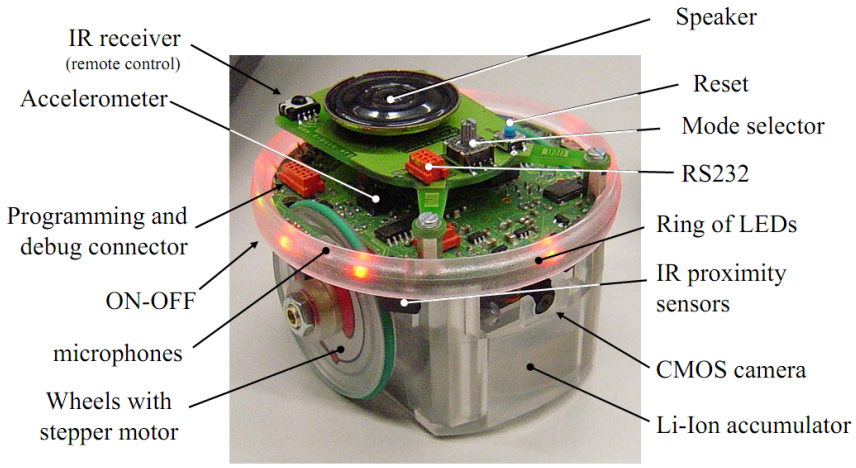

Figure 1: E-puck robot

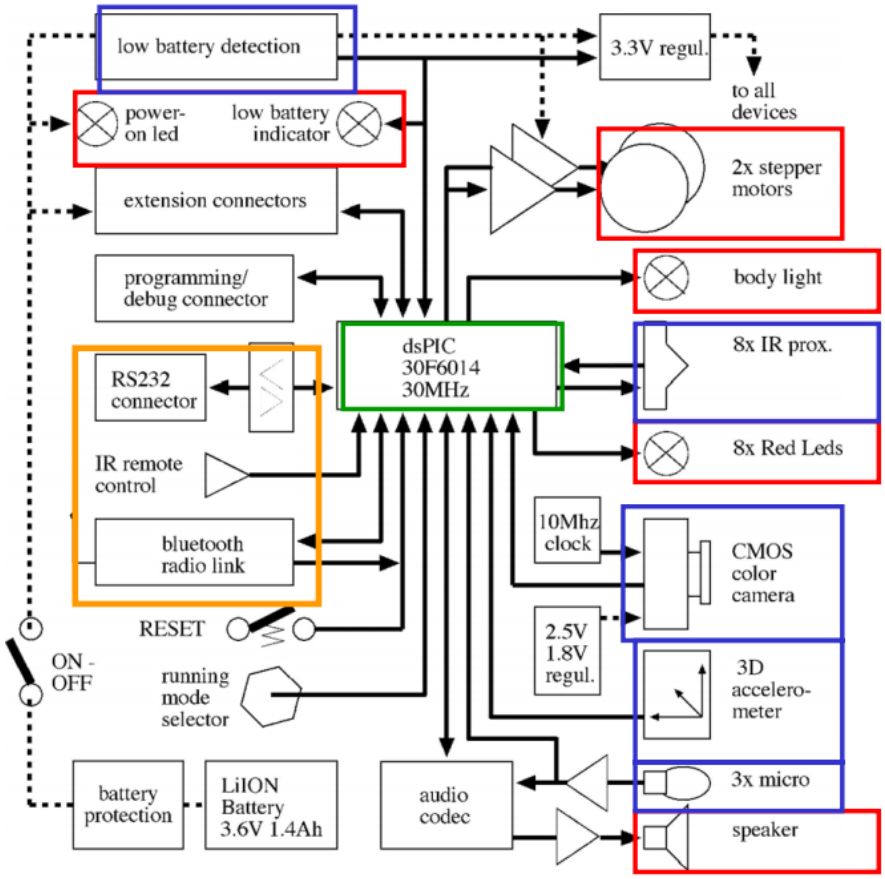

Figure 2: The outline of the electronic of the e-puck

- 8x IR proximity: They measure the closeness of obstacles or the intensity of the ambient light. These are placed around the body.
- CMOS colour camera: A colour CMOS camera with a resolution of 640 x 480 pixels is placed in front of the robot. The size of acquisition is limited by the memory size of the dsPIC and the rate is limited by its processing power.
- 3D accelerometer: Measures the acceleration produced by its own motion and to measure

the inclination of the e-puck.

- 3x micro: Localize the source of the sound by triangulation.

The e-puck provides various actuators, identified as red (Figure 2) to send orders:

- Power-on: It is located under the chip and next to the right wheel.
- 2x stepper motors: They control the motion of the wheels. They receive a value from -1000 till 1000. A negative value moves the wheel backward and a positive value moves forward.
- Body light: A set of lights placed in the transparent body offer the possibility to interact with the user.
- 8x red lights: They are placed around the e-puck and are emitting diodes (LED). They provide a visual interface with the user and could be useful for other e-pucks which are using the camera.
- Speaker: Connected to an audio codec. Offers another possibility to interact with the user.

The e-puck has two communication possibilities, identified as orange (Figure 2) to interact with devices:

- RS232 connector: Serial interface to communicate with a computer using a cable.
- Bluetooth radio link: Wireless communication with a computer or with up to 7 other e-pucks.

Software

There are different software available for e-puck and these will be defined below:

- Bootloader: Permits transfer programs in format file .hex, previously compiled using particular software recommended by `www.e-puck.org`, whose name is MPLAB and allows programs to be created using C as a language, to compile them and create a .hex file. The recommended name to upload .hex files to the robot is "Tiny Bootloader".
- Standard library: A collection of modules for using the different sensors, actuators as well as communications to facilitate the use of the e-puck hardware is provided.
- Simulation: There are various simulators of the e-puck, for instance Webots, is a development environment used to model, program and simulate mobile robots [3]. The fourth step (Figure 3) would be to transfer the program which Webots generates to e-puck using software to upload the file or files. One of the advantages of using this environment is the possibility to program the behaviour for the robot in different languages as (python, c# or Java) and then automatically translate to C.
- Monitor: To check the sensors and actuators of the e-puck, there are two possibilities; one is .exe program, which runs in Windows platforms and the other, a program that runs inside of Matlab. These programs help to detect if there is any sensor or actuator which does not work.

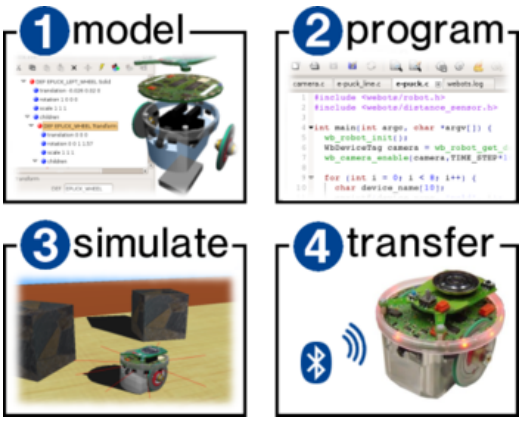

Figure 3: Webots steps to develop

Interface with the robot

The robot contains several devices to interact with as can be seen in (Figure 1) to identify each one. The different devices to interact will be explained below:

- There are three connectors: in-circuit debugger, to program flash memory and to debug code.
- ON – OFF: This switch permits powering on and off. It cannot be seen in (Figure 1) but is located under the chip.
- Reset: This button restarts the program running in the robot when robot is power on.
- Mode selector: The robot has sixteen positions which can be used. In each position a different program can be executed. Remember, when position is changed, there is need to press Reset button.

Mechanisms and useful theories

The robot e-puck has a differential configuration. It consists of two drive wheels mounted on a common axis, and each wheel can independently move either forward or backward [4]. There

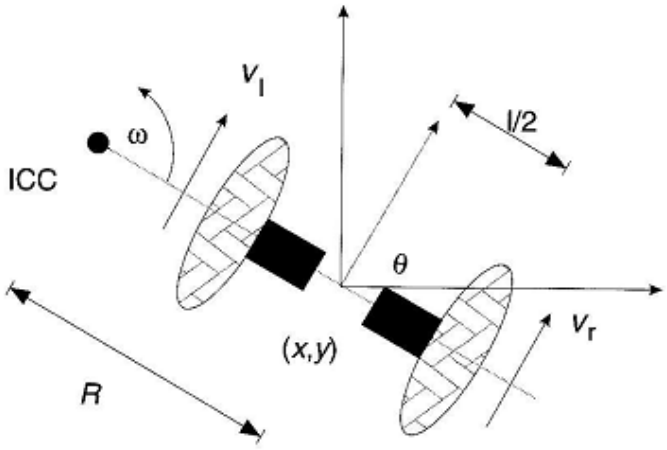

Figure 4: Differential drive kinematics

are 2 steppers motors with gear reduction. A stepper motor can move in accurate angular

increments known as steps in response to the application of digital pulses to an electric drive circuit from a digital controller. The number and rate of the pulses control the position and speed of the motor shaft. E-puck motors have 20 steps per revolution.

The wheels diameter is about 41 mm and they are separated by distance of 53 mm. The maximum speed of the wheels is 1000 steps per second, which corresponds to one wheel revolution per second [21].

By varying the speed of the two wheels, the trajectories that the robot takes can be varied. There are three interesting cases:

1. If, $Vr = Vl$ robot moves in a straight line.
2. If, $Vl = -Vr$ robot turns clockwise. Whether $Vr = -Vl$, robot turns counter clockwise.
3. If, $Vl = 0$ there is a rotation about the left wheel. Whether $Vr = 0$, there is a rotation about the right wheel.

As robot has a differential drive configuration, imposes non-holonomic constrains on establishing its position. It means that robot cannot move laterally along its shaft. So, cannot simply specify an arbitrary robot position and find the speed needed to reach that position.

To control the robot and reach a given configuration $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$ there is need to know the inverse kinematic:

$$x(t) = \int_0^t \upsilon(t) cos[\theta(t)] \, \mathrm{d}t \tag{1}$$

$$y(t) = \int_0^t \upsilon(t) sin[\theta(t)] \, \mathrm{d}t \tag{2}$$

$$\theta(t) = \int_0^t \omega(t) \, \mathrm{d}t \tag{3}$$

where υ refers to linear speed and ω is the angular speed. To apply this principle for a motion in a straight line, the (Equation 4) shows how to reach the position:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + \upsilon cos(\theta) \, \mathrm{d}t \\ y + \upsilon sin(\theta) \, \mathrm{d}t \\ \theta \end{bmatrix} \tag{4}$$

If the robot rotates over itself (Equation 5) is used which allows the position to be got:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y' \\ \frac{2\upsilon \, \mathrm{d}t}{l} \end{bmatrix} \tag{5}$$

The variable l refers to diameter of the robot axis, as can be seen in (Figure 4). Below, the (Table 1) represents a summary of useful data for solving any problem related with the project.

Table 1: Variables for e-puck

Variable	Value	Meaning
l	53	Distance in mm between the wheels.
d	41	Diameter in mm per each wheel.
s	1000	Speed in steps per second.
w	2π	Angular speed in radians per second, per each wheel
Θ	$d\pi$	Maximum speed in mm per second.
Vl	$\frac{\iota}{s}\Theta$	Speed for left wheel giving ι steps, where ι is an integer value [-1000,1000].
Vr	$\frac{\iota}{s}\Theta$	Speed for right wheel giving ι steps, where ι is an integer value [-1000,1000].

Design discussion

In this chapter has been explained the different alternatives studied to reach each requirement. Each alternative as well as its advantages and disadvantages are explained. Regarding the environment to simulate and the robot selected (e-puck) are not discussed nothing since these requirements are closed without possible changes.

Basic actions approaches

"Basic Actions" explained in the previous chapter, involve know where is located the robot every-time. Localisation involves determine the Cartesian coordinates (x, y) and the orientation θ known as the state $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$. Three alternatives are studied in this project:

Basic actions using Odometry

Odometry, also known as Dead Reckoning, is the measurement of wheel rotation as a function of time[2]. If the two wheels are joined to a common axle, to estimate position in time t the previous position as well as the linear speed v and angular speed w are used, the (Equation 1,2,3) show how get the position. Thus, the change of position during time, is known through encoders of the wheels.

It is considered as the first approach to know an estimation about robot position $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$. The advantages of this approach are:

- It requires only the use of encoders of the motors to get the position and there not need to use other sensors for estimating of position.
- If length paths are short, it provides good accuracy of position estimation.
- It is compatible with other positioning approaches. Hence, this approach combined with other allows get better results.

The disadvantages of this are:

- Encoders return discretized values, losing precision due to the motors are represented as continuous function really.
- The estimation of position can be better or worse depending on floor type where robots are located.
- The wheels diameter inequalities influence in the estimation of position.
- If the wheels produce slippage, a loss of position is got.

Basic actions using Kalman Filter

The Kalman Filter (KF) provides an efficient computational tool to estimate the state of a process, minimizing the mean squared error [19]. KF has two steps:

- The **update** uses the dynamical model of the system (forward kinematics in e-puck), requiring an initial position to start.
- The **predictor** uses a sensor model, such as the error distribution calibrated from its sensors, for getting an improved estimation.

Hence, KF updates the state of the system, that are the position for the robot as well as its variance.

$$\begin{aligned} x_k = Ax_k - 1 + Bu_k - 1 + w_k - 1 \\ z_k = Hx_k + v_k \end{aligned} \tag{6}$$

where $x \in \Re^n$ is the state vector, $u \in \Re^u$ is the input vector and $z \in \Re^m$ is the measurement vector. w_k,v_k represent process and measurement noises at time k. They are considered to have an independent, white, normal probability distribution with zero mean. The exact values of w_k and v_k at time k are normally unknown but it is assumed the knowledge of the covariance matrices, Q_k and R_k respectively(assumed to be constant). The KF aim is to estimate the state x_k of (Equation 6) based on the knowledge of the system dynamics (linear model), the covariance matrices (disturbance characteristics) and the availability of the noisy measurements z_k. More details about how works this approach can be found in [19]. The advantages of this approach are:

- It is obtained better accuracy than using odometry since with this approach are fused various sensors for getting better estimation of position.
- It is possible to apply this filter for different purposes like speed of the wheels or speed of the robot.
- There are more extensions for this filter that provide better estimation for robot position, for instance extended KF.

The disadvantages of this approach are:

- If the platform where the filter is implemented has limited resources, as e-puck robot, it is difficult develop the approach due to resources limitations.
- Calibration of the wheels is need to get good estimation of position, that are, measure the diameter for each wheel and remove slight differences in wheels speed.

Basic actions using Particle Filter

Particle Filter (PF) is an approach that can be used to estimate position of the robot. The main objective of PF is to "track" a variable of interest as it evolves over time [10]. The variable of interest has multiples copies (particles) and each copy has a weight which corresponds with quality of that specific particle. Hence, the value of interest variable is got by the weighted sum of all particles.

The PF algorithm has three phases:

1. **Prediction:** It uses a model in order to simulate the effect an action has on the set of particles with appropriate noise added. For instance, to estimate robot position, odometry can be used, adding after a noise model for motions performed. According authors [10] if an arbitrary motion is performed as a rotation followed by a translation, is needed to apply a noise model separately, because these motions are independent.

2. **Update:** It uses information obtained from sensors like (proximity sensors, laser, camera) to update the particle weights in order to accurately describe the moving probability density function (pdf) of the robot.

3. **Re-sampling:** This phase removes particles with small weights since these particles become to small to contribute to the (pdf) of the moving robot.

There are three methods of evaluation to obtain an estimate of the position, according authors [10]:

- *Weighted mean:* $\begin{bmatrix} x & y & \theta \end{bmatrix}^T = \sum_{j=1}^{M} w_j x_j$, where M is the number of particles sampled, w_j is the weight associated with the particle and x_j refers to particle j.

- *Best particle:* It selects the particle with maximum weight associated. $\begin{bmatrix} x & y & \theta \end{bmatrix}^T = max(w_j) : j = 1...M$.

- *Robust mean:* A weighted mean in a small windows around the best particle. It is the method most computationally expensive.

The advantages of this approach are:

- If sensors like camera o laser are used, allows get a global estimation of the position, being more reliable than others approaches which obtain only a local approximation of the position.

- If the number of samples are increased, allows get better estimation of position.

- There is not restrictions in model, so can be applied to non-Gaussian models, hierarchical models or even others.

and the disadvantages are:

- If the number of samples are increased, it can be inefficient computationally or even inaccessible for e-puck.

- It can be hard find programming errors due to different phases and their associated calculations.

Protocol avoidance approaches

Protocol avoidance, explained in the previous chapter, involves avoid crashing with the walls and the other robots located in the environment. There are two alternatives for explaining these requirements:

Protocol avoidance using Proximal (reactive)

In this approach the robot sensors control the speed of the motors and these are changed according the value got by sensors. To implement this approach a matrix is used normally where rows represent each sensor and columns are associated to each motor. Thus, the speed is the sum of rows associated with the column that represents the motor. For instance, e-puck robot has eight proximity sensors, each one would be associated in a row and it has two motors where would be represented in two columns [9].

The weights of the matrix are determined empirically and they depend on the problem to solve. The (Equation 7) estimates the speed of the motors:

$$speed_{motors} = \sum_{\substack{0<i<m \\ 0<j<n}} matrix(i,j) \cdot (1.0–(sensorValue(i)/v)); \tag{7}$$

where i is weight associated for each sensor, j is each motor, $sensorValue(i)$ represents the value got from the sensor and v is a value estimated empirically to normalize the sensor value, for instance could be 512. The advantages of this approach are:

- There is not need to establish actions to do when an obstacle is detected since it tries to avoid obstacles, varying motors speed.
- The motions for avoiding obstacles are less abrupt since the speed is controlled by proximity sensors.

The disadvantages of this approach are:

- It is needed a very good calibration of the sensors for avoiding behaviours in motors speed unexpected .
- It is needed find out weights of the matrix, changing depending on problem.
- Whether the robot is crossing over a narrow passage, it will remain in this situation probably.
- If the robot has objects in front of it, as well as in its left and right, it will remain in this situation probably. This type of scenario is known as "U-shaped".

Protocol avoidance using Distal (rule-based)

In this approach is established a threshold for each sensor and when this value is overcome, it is decided the actions to do [9]. For instance, the next code provides an example combining the sum of two sensors:

$$sensorValue(i) + sensorValue(j) > k\{action\ or\ actions\ to\ do\} \tag{8}$$

where k is a value (threshold) established empirically, although exist other examples for determining the actions to do. The advantages of this approach are:

- There is flexibility to establish the behaviour which performs the robot, allowing to choose action or actions to do depending on the criteria satisfied.
- It is easy to detect possible mistakes in code since the actions can be separated depending on the sensor.

The disadvantages of this approach are:

- Whether the robot is crossing over a narrow passage, it will remain in this situation probably.
- If the robot has objects in front of it, as well as in its left and right, it will remain in this situation probably. This type of scenario is known as "U-shaped".

Concurrence approaches

Concurrence, explained in the previous chapter, involves run a program which controls robots in one computer simultaneously. Moreover, there is another restriction to satisfy, the program must run in Java. So, in Java exists thread approach which allows an application to have multiple threads of execution running concurrently [13].

A Thread has a priority and depending on its priority will be executed before or after another. There are two ways to create a new thread of execution:

Concurrence using Thread

It must be declared a subclass which extends from Thread and it should modify the run method. The principal disadvantage using this way is that Java language does not allow more than one class to extend, providing less flexibility. (Listing 1) provides an example which uses this approach. Firstly, there is need to create an object myThread, and after, call start method for starting to run the Thread.

Listing 1: Java. Class extends Thread

```
1 class myThread extends Thread{
2         int value;
3         myThread(int value){
4                 this.value = value;
5         }
6         public void run(){
7                 //Override this method
8         }
9 }
```

Concurrence using Runnable

It must be declared a class that implements the Runnable interface. This type of approach provides more flexibility than previous approach, above mentioned, because in Java language it is possible implements various classes. (Listing 2) provides an example which uses this approach. Firstly, there is need to create an object myRunnable, and after, call start method for starting to run the Thread.

Listing 2: Java. Class implements Runnable

```
1 class myRunnable implements Runnable{
2         int value;
3         myRunnable(int value){
4                 this.value = value;
5         }
6         public void run(){
7                 //Override this method
8         }
9 }
```

Implementation issues (excluding programs listings)

In this chapter it is explained the structure of system in two sides, Java and C. Java is the language chosen to achieve requirements like concurrence and C is the language supported by e-puck. Further, the approach chosen for each requirement as well as its code developed are explained.

Structure of the system (Java side)

Java has been the language chosen since one of the aims, explained in "Introduction", requires a communication between this language and e-puck robots. Thus, following the principles of Software Engineering a class diagram has been created to provide a general vision about the code implemented in Java. Bellow, each class in (Figure 5) will be explained:

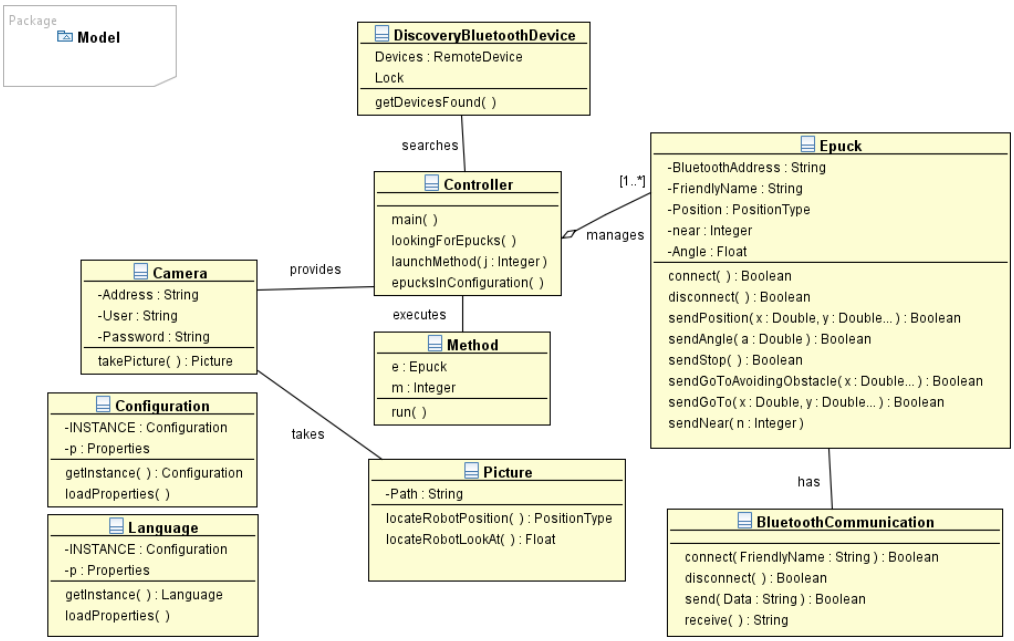

Figure 5: Class diagram (Java side)

- **DiscoveryBluetoothDevice:** This class searches bluetooth devices installed in the machine where it is running. It implements *DiscoveryListener* which allows to receive device discovery and service discovery events [1]. It has two attributes, *Devices* (a vector of *RemoteDevice*) and *Lock* (an object to lock the process until inquiry is completed). The method *getDevicesFound()* returns a vector of *RemoteDevice* found in the computer or null if an error happens.

- **Controller:** This class is the entry point to the system. It has four methods,
  - main() it is the method executed when program starts. It allows to choose between different actions (looking for e-pucks, proximity sensors calibration...).
  - *lookingForEpucks()* it shows in screen, friendly name and bluetooth address for each e-puck installed in the machine.
  - *launchMethod()* it launches the method selected in *main()* method.
  - *epucksInConfiguration()* it reads each e-puck and its attributes in a configuration file.
- **Epuck:** This class is a abstraction of an e-puck. It has five attributes,
  - *BluetoothAddress:* it is an unique identifier for each e-puck. It is used to connect with the e-puck. For instance, e-puck 2479 has *1000E8AD6AC4* as identifier.
  - *FriendlyName:* It is the name for each e-puck and it is different for each one. For instance, e-puck 2479 has *e-puck_2479* as friendly name.
  - *Position:* It represents the position (x, y) that robot has.
  - *Angle:* It represents the angle θ that robot is facing.
  - *Near:* It represents the threshold for taking into account that there is an object near.

  and has ten significant methods,
  - *connect()* it tries to connect with the e-puck, through its *BluetoothAddress* identifier.
  - *disconnect()* it tries to disconnect the e-puck.
  - *sendPosition()* it sends a position (x, y) where the robot is located. It is useful when robot loses the reference position.
  - *sendAngle()* it sends an angle θ where the robot is facing. It is useful when robot loses the reference angle.
  - *sendStop()* it sends a signal to stop the process running in the robot.
  - *sendGoToAvoidingObstacle()* it sends a position to reach, taking into account avoid crashing with obstacles.
  - *sendGoTo()* it sends a position to reach.
  - *sendNear()* it sends a new threshold. It is useful when ambient light is changing.
  - *sendOverallTime()* it sends this order, for receiving the overall time elapsed of the orders that (Algorithm 1) executes.
  - *sendObstaclesDetected()* it sends this order, for receiving the number of obstacles detected for orders that (Algorithm 1) executes.
- **Method:** it is the class which allows to connect various e-puck simultaneously. It receives an *Epuck* object as well as an integer m that determines the method to run. It has various methods but *run()* is the most important since decides the method to execute concurrently.
- **BluetoothCommunication:** It is the class that implements the communication between Java and e-puck. It has four methods *connect()*, *disconnect()*, *send()* and *receive()*. They are explained in "Communication Java and E-puck".
- **Camera:** This class interacts with an IP Camera through *http*. It has three attributes,

  - *Address:* It is an *http* address where the camera is installed.
  - *User:* I is the user name to connect with the camera.
  - *Password:* It is the correct password to connect with the camera.

  and has one method, *takePicture()* that takes a picture of the environment where camera is pointing.

- **Picture:** it represents an abstraction of an image. It has one attribute, *path* that refers to path where the picture is stored. It has two methods,
  - *locateRobotPosition()* it finds the coordinates (x, y) where the robot is located in the picture.
  - *locateRobotLookAt()* it finds the angle where the robot is facing in the picture.
- **Configuration:** This class implements *Singleton pattern* to avoid more than one instance can be run simultaneously when the program is executing. It has two attributes *INSTANCE* that references the class (itself) and p that represents the different properties, reading them from *CONFIGURATION.txt* file. These properties allow to update the different attributes of some classes, mentioned previously. Properties is a class which represents a persistent set of properties, more information [12].
- **Language:** This class has an important attribute p which represents the different properties, reading them from *english.txt* and these properties represent messages that application produces. It can be useful if it is required translate the messages to another language.

Structure of the system (C side)

E-puck robots can be programmed using C language and through the standard library, the sensors, the actuators as well as the communications are possible to control. It is recommended to read the standard library documentation that can be downloaded in the official site of e-puck [22] and it provides good explanations about the different files integrated and the data structures which has.

In order to show the structure of the system in e-puck side a list of functionalities, that robot have implemented, has been created. Below, each functionality, the files which contains and the different functions developed will be explained:

- **runbrw:** It is the controller for the system and every action, that robot performs, is managed on this functionality. This contains two files, *runbrw.h* and *runbrw.c*. Also it has these functions:
  - *void run_brw():* It is the main function for the project. It always is waiting until an order is coming for doing the correct action associated with order.
  - *void protocol(char *data, int length):* It is the **order interpreter** which decides the action to do depending on the order received.
- **runbrw_bluetooth:** It allows to send and receive data through bluetooth. This contains two files, *runbrw_bluetooth.h* and *runbrw_bluetooth.c*. Its functions associated are:
  - *char * receive(int *length)*
  - *void send(char *output, int length)*

these are explained in "Communication Java and E-puck" section.

- **runbrw_motion:** It implements every action related with motions and it is the functionality which implements the simulation associated with Escherichia Coli. This contains two files, *runbrw_motion.h* and *runbrw_motion.c*. Also it has these functions:
  - *void go_to(float x, float y):* it allows to reach (x, y) from current coordinates without obstacles avoidance.
  - *int go_to_avoiding_obstacles(float x, float y):* it tries to reach (x, y) from current coordinates taking into account possible obstacles in the path. If there is an obstacle on coordinates objective it cannot achieve (x, y), stopping as near as possible.
  - *void odometry_clockwise():* it is explained in "Odometry as approach used" section.
  - *void odometry_counterclockwise():* it is explained in "Odometry as approach used" section.
  - *void odometry_forward():* it is explained in "Odometry as approach used" section.
  - *void turn_angle(double x):* it is explained in "Distal (rule-based) as approach used" section.
  - *void forward(double x,double y):* it is explained in "Distal (rule-based) as approach used" section.
  - *void forward_avoidance():* it is explained in "Distal (rule-based) as approach used" section.
- **runbrw_protocol:** This functionality implements regular expressions to convert string data coming from Java and it adapts those in its correct format. This contains two files, *runbrw_protocol.h* and *runbrw_protocol.c*. Its functions are:
  - *void regExp(char *data,int length,double *x,double *y):* it is explained in "Communication Java and E-puck" section.
  - *void regExp2(char *data,int length,double *x):* it is explained in "Communication Java and E-puck" section.
  - *void regExp3(char *data,int length,int *n):* it is explained in "Communication Java and E-puck" section.
- **runbrw_proximity:** it allows read data coming from proximity sensors and it determines the behaviour to do. This contains two files *runbrw_proximity.h* and *runbrw_proximity.c* and its function programmed is *int something_near(void)* which allows to detect an obstacle and where it is located. More details can be found in "Distal (rule-based) as approach used" section.
- **runbrw_utility:** it provides support for different utility functions, allowing have a good structure of the code programmed and avoiding duplicate code in parts different of the program. The files that it contains are *runbrw_utility.h* and *runbrw_utility.c*. Also the functions implemented are:
  - *void flicker_leds(int seconds):* it allows to the eight led blink during the seconds given.
  - *double get_angle(double x, double y, double x_, double y_):* it calculates the angle between points given. Further details in "Distal (rule-based) as approach used" section.

- *double get_angle_calibrated(double x, double y, double x_, double y_):* it calculates the angle between points given, changing the range values. More details in "Distal (rule-based) as approach used" section.
- *double get_distance(double x, double y, double x_, double y_):* it obtains the Euclidean distance for the points given.
- *void time_pause():* it increments a variable of time one millisecond.
- *void pause_brw(int seconds):* it pauses the process which calls to this function.

- **runbrw_variables:** it stores every useful constant variable and it has all references for global variables that can be used in every part of the program. It contains only one file *runbrw_variables.h* and it has defined inside twenty-one variables (constants or global). For instance, *MOTOR_SPEED* refers to the motors speed when robot moves forward, whilst MOTOR_SPEED_ANGLE refers to the motors speed when robot turns over itself. All of these variables are capitalized.

Odometry as approach used

Odometry has been the approach chosen for reaching the requirements related with "Basic Actions". Generally the position of a robot is represented by the vector $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$. For a differential-drive robot the position can be estimated starting from a known position by integrating the movement (Equation 1,2,3). Thus, using information about the encoders during interval t, it is possible to know the distance travelled by each wheel. Below, the following equations [11] allow to get position at time t:

$$\Delta x = \Delta s \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right) \tag{9}$$

$$\Delta y = \Delta s \cdot \sin\left(\theta + \frac{\Delta\theta}{2}\right) \tag{10}$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{l} \tag{11}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{12}$$

$$\Delta s_l = \frac{\Theta \cdot steps\_elapsed\_left}{s} \tag{13}$$

$$\Delta s_r = \frac{\Theta \cdot steps\_elapsed\_right}{s} \tag{14}$$

where the incremental travel distances $(\Delta x, \Delta y, \Delta\theta)$ are the path travelled in the last sampling interval t. Δs_l and Δs_r are travelled distances for the right and left wheel respectively and (Θ, l) are described in (Table 1). Hence, the current position in interval t is:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{15}$$

How have equations been implemented in e-puck?

The most important thing to get a maximum precision is to determine the sample period which has been established to *1 ms* because in one second the robot runs *1000 steps*, thus in *1 ms* robot runs *1 step* theoretically. Then, using **agenda** which consists in establish a function

to run every *t ms*, update the position of the robot every *1 ms* has been possible. This agenda is implemented in standard library coming with e-puck.

(Listing 5) updates each millisecond Δx and Δy, reading the number of elapsed steps in left and right wheel for the last sample period, the distance travelled for left and right wheel is calculated using (Equation 13,14) and (x, y) is updated finally.

(Listing 6,7) update $\Delta\theta$ either counterclockwise or clockwise, reading the number of elapsed steps per each wheel, after the distance travelled is calculated in each wheel using (Equation 13,14) and θ is updated finally. Though, there are two functions to solve the negative or positive angle, exists only one difference between them, that is, the negative increment is on left for anticlockwise and is on right for clockwise.

It is assumed that Θ in (Equation 13,14) is a constant value associated with the distance travelled in one second and it is similar in wheels left and right. Also s in (Equation 13,14) which corresponds with the speed in steps and it is the same for both wheels. As can be seen in these functions the elapsed time to achieve every distance can vary depending on the speed established in motors, thus high values for motor speed allow to reach the objective soon than low values.

Distal (rule-based) as approach used

In order to achieve the requirements associated with "Protocol avoidance" has been chosen approach Distal since it is possible manage better the actions that robot need to do when an object is detected.

Hence, the values of proximity sensors 0 and 1 are combined to determine an obstacle on right and the values of proximity sensors 6 and 7 are merged to detect obstacle on left. Thus, using an extension of (Equation 8) it is possible to apply approach Distal and later the actions decided to do. The (Algorithm 1) written in pseudocode shows the behaviour for achieving avoid obstacles. Below, the actions associated with this algorithm will be explained:

- ***turn_angle($\theta^{'}$).*** As can be seen in (Algorithm 1.2) and supposing that robot is facing 0 degrees, the robot turns over itself $\theta^{'}$ radians. To obtain $\theta^{'}$, a trigonometric function in (Equation 16) is used,

$$angle = \arctan\left(\frac{y^{'} - y}{x^{'} - x}\right) \tag{16}$$

  but this function returns a value between $[\frac{\pi}{2}, \frac{-\pi}{2}]$ and would involve that robot moves forward and backward, complicating avoidance protocol. So (Listing 3), a customized function, returns a value between $[\pi, -\pi]$ preventing the robot moves backward.

- ***forward($x^{'}$,$y^{'}$).*** As can be seen in (Algorithm 1.4), this action allows to the robot moving forward from (x, y) to $(x^{'}, y^{'})$. When this function detects an obstacle, motors speed are updated to 0 and the source of the obstacle is notified. The source can be, an obstacle located or left either right. (Listing 4) determines where is located the source of the obstacle and counts the number of times that something is detected.

- ***turn_anticlockwise.*** As can be seen in (Algorithm 1.9), this action is executed when an obstacle on right is detected, that is, if sensors 0 and 1 detect an obstacle. Hence, the action is turn the robot θ'' anticlockwise and the value selected is $\frac{\pi}{4} radians$ for this project.

- ***turn_clockwise.*** As can be seen in (Algorithm 1.12), this task is executed if and only if, an obstacle on left is detected, that is, whether sensors 6 and 7 detect an obstacle. Thus, the action is turn robot θ'' clockwise and the value selected is $\frac{-\pi}{4} radians$ for this project.

Algorithm 1 Obstacles avoidance algorithm

```
1: while ((x ≠ x' ∨ y ≠ y') ∧ try < OPPORTUNITY) do
2:     θ = turn_angle(θ')
3:
4:     forward(x',y')
5:
6:     if OBSTACLE then
7:         while OBSTACLE do
8:             if RIGHT then
9:                 θ = turn_angle(θ'')
10:            end if
11:            if LEFT then
12:                θ = turn_angle(−1 · θ'')
13:            end if
14:            forward_avoidance()
15:        end while
16:        Δtry
17:    else
18:        x = x' y = y'
19:    end if
20:
21:    θ = turn_angle(0)
22: end while
```

- ***forward_avoidance***. As can be seen in (Algorithm 1.14), this task moves the robot forward during k steps established empirically for avoiding the obstacle detected but during this action it is possible to detect other obstacle, so it is need going back (Algorithm 1.7).

- ***turn_angle(0)***. As can be seen in (Algorithm 1.21), this action moves the robot to 0 degrees. It always involves to know where the robot is facing for turning to 0 degrees. It is need to do this action since (Equation 16) assumes that robot is facing 0 degrees.

Therefore, the algorithm finishes when (Algorithm 1.1) is satisfied, otherwise remains in this algorithm. But, what is up if the robot tries to reach coordinates and an obstacle is there? Well, to simplify this behaviour coordinates objectives are tried to reach during *try* times and if it is not possible to achieve them, the robot stops as near as possible from aim coordinates.

Summarizing, the principal actions, that algorithm does, has been explained because various actions are done in different parts when an obstacle is detected. Firstly, the source of the obstacle (right or left) is detected, secondly turn clockwise either anticlockwise depending on where the source is located, third move it away from obstacle and finally the new angle for coordinates that are away from obstacle detected is calculated.

Thread as way used

To achieve the requirements associated with "Concurrence", Thread way has been chosen, although is less flexible than Runnable it can be selected because it is not need to extend from another class for this project.

(Listing 8) shows the class which implements this approach. This class receives an object e-puck as well as an integer for deciding which method it is need to execute. Moreover, to allow

the concurrence *run()* method has been overwritten and it is executed per each e-puck switch on and installed in computer properly.

Communication Java and E-puck

One of the most important things that has been developed for this project is the communication needed between e-puck and Java. Thus, after hard research in different sources it is concluded that nobody has developed an Application programming interface (API) to communicate Java language with e-puck. Hence, a challenge with success has been the development of a protocol to allow this communication.

This communication is established by the program in Java which is a Bluetooth client in each e-puck. In this communication bytes codified in ASCII are sent, then each e-puck waits till the end of the message is reached, posteriorly these bytes are decoded and finally the action to do is decided, sending a reply to Java program which is waiting for sending another order.

The ***message*** always starts with *STX*, start of text whose representation in ASCII is the number *2*. Posteriorly, a letter representing the action to do followed by the parameters for this action and the end of the communication *ETX*, end of text, whose representation in ASCII is the number *3*. Below, the (Table 2) shows the different messages for sending to each e-puck:

Table 2: Java messages for sending to each e-puck

Message	**Meaning**
sendPosition(double x, double y)	Updates coordinates in e-puck.
sendAngle(double a)	Updates angle in e-puck.
sendGoTo(double x, double y)	coordinates objective to reach.
sendGoToAvoidingObstacle(double x, double y)	coordinates objective to reach, avoiding obstacles.
sendNear(int n)	Updates threshold in proximity sensors.
sendStop()	Interrupts whatever action running in the robot.
sendOverallTime()	Allows get elapsed time over actions "goto" done.
sendObstaclesDetected()	Allows get number of things detected over actions "goto" done.

This ***protocol*** has the following actions (in Java side):

- **Connection:** It allows to connect with e-puck. First, it tries to connect with the e-puck passing its "bluetoothAdress". Then if there is success it is established a channel for input and other for output, returning true or false whether something happens. (Listing 9) shows how to do that.
- **Disconnection:** it allows to disconnect with e-puck. It tries to close channels and connection, returning true or false if something happens. (Listing 10) shows how to do that.
- **Send:** This method receives a string of characters. First, it writes *2 (STX in ASCII)*, posteriorly converts each character in ASCII and it writes *3 (ETX in ASCII)* finally. (Listing 11) shows these explanations.

- **Receive:** This method locks the program till something is coming. First, it removes *2 (STX in ASCII)*, then creates a string with characters received and finishes when *3 (ETX in ASCII)* is coming. (Listing 12) shows these explanations.

and has the following actions (in e-puck side):

- **Send:** It allows to send a message from e-puck to Java, creating an string that has *2 (STX in ASCII)* first, then the message and *3 (ETX in ASCII)* finally. (Listing 13) shows how to do that.

- **Receive:** This function is waiting till something is coming. If the character received is *2 (STX in ASCII)* or *3 (ETX in ASCII)* it is discarded but when *3 (ETX in ASCII)* is received, the function ends, returning the string and its length. (Listing 14) shows how to do that. To have a good feedback, body light is switch on when the robot is waiting data.

The encoding and decoding are done in both sides. Thus, to ***encoding*** the data each character of the data is converted to ASCII. For instance to convert the message *sendPosition(22.50, 14.36)* as a string, it is *"P22.50;14.36"* and its ASCII is in (Table 3).

Table 3: Example encoding message from Java

Character	**ASCII**
STX	2
P	80
2	50
2	50
.	46
5	53
0	48
;	59
1	49
4	52
.	46
3	51
6	54
ETX	3

To ***decoding*** each message the creation of different regular expressions has been needed since the string received can have various numbers inside and it is need to separate each one and later convert to double or integer depending on the first letter received in the string. (Listing 15) provides the code needed to get two doubles, the delimiter *";"* allows to separate each one. This regular expression is used for messages *sendPosition(double x, double y), sendGoTo(double x, double y), sendGoToAvoidingObstacle(double x, double y)*, whilst if only is coming a double (Listing 16) provides it and whether is coming an integer (Listing 17) offers it. Both are for messages *sendAngle(double a), sendNear(int n)* respectively.

To ***decide*** the action to do, the first letter in the string decoded allows to determine the function for executing in the e-puck. Thus, this type of communication established between Java and e-puck is like an "orders interpreter", because each e-puck executes a determinate function depending on the first letter of this string received. (Listing 18) provides the method which determine the action to do.

Summarizing, to establish communication between Java and e-puck has been created:

1. A determinate format for messages.
2. Connection, disconnection, send and receive methods in Java.
3. Send and receive methods in e-puck.
4. Encoding and decoding messages in both sides, due to channel accepts bytes only.
5. Regular expressions to allow e-puck understands what is coming.
6. Order interpreter to decide which action execute in e-puck.

Improvements for resource limitations of e-puck

E-puck robot has limitations of memory, so the amount of memory which it has is 8KB. Therefore, it is necessary to take into account this amount of memory for every code programmed otherwise it can be locked sometimes. Thus, there are two examples performed in this project trying to avoid full memory of e-puck. These examples are:

- **Release memory:** This action is done in last line at principal loop of the program since a pointer of size $TAMBUFFER$ is created every time in a function that is waiting for everything coming through Bluetooth. Thus, once order interpreter sends the reply to Java, the memory created is released, avoiding getting it filled by useless data. Line 11 of (Listing 19) shows how this action is done.
- **Management of agenda:** This action allows a good management of agenda to be done. This is an approach developed in standard library of e-puck which provides the possibility of executing functions at certain times. For instance, to obtain estimation of position, wheels increments are calculated each millisecond but this function is not always called, only when robot speed is different from 0. Thus, if the agenda for this function is activated and then destroyed when wheels speed are 0, the function for destroying agenda does not release the memory, so in a short time the memory of the e-puck is full. Thus, the behaviour for a good management of agenda is:

  1. Create an agenda (once) for each function which needs to update a particular action in a certain time.
  2. Pause agenda when the function is not needed for updating the action.
  3. Restart agenda when the function is needed for updating the action.

  (Listing 20) shows the code for the example explained previously.

Analysis of system performance

In this chapter is presented two experiments for trying to achieve every requirement imposed in the project. The first experiment executes one e-puck and the second executes three e-pucks concurrently. For each one the scenario chosen is detailed as well as the result obtained. Experiments execute orders coming from the program which executes BRW really. This program has been developed by authors [8] and produces a file which the format is *friendy_name;x_coordinate;y_coordinate*. An example line would be *e-puck_2479;39.0;95.0*.

For overall time is summing elapsed time for performing turns and motions forward without counting pause and communication times.

First experiment

The first experiment executes BRW in one robot trying to achieve "Basic Actions".

First experiment - Scenario

The scenario for this experiment has the following characteristics:

- The start coordinates are $(4.00, 3.50)$.
- It receives twenty-five orders.
- The speed to move forward the robot is 800 steps.
- The speed to turn the robot is 1000 steps.
- The environment where the robot is placed is an enclosed surface with green carpet floor.
- Each order is received from computer through Bluetooth.

First experiment - Results

The blue line (Figure 6) corresponds to path which one robot performs in simulation mode and the red line (Figure 6) corresponds to path which does $e-puck\_2479$ robot. (Table 4) shows the values represented in (Figure 6), where "(x, y) BRW" refers to coordinates executing program simulation in Java, whilst "(x, y) BRW robot" refers to coordinates reached at *e-puck_2479* robot. The **average absolute error in x** is 2.752 and **average absolute error in y** is 2.988, the **overall time** for executing these orders is 54.914154 seconds. It is not produced slippage in wheels during this experiment and the communication for sending works without errors. The output created by Java is shown in (Listing 21), the video executing this experiment can be downloaded from [17].

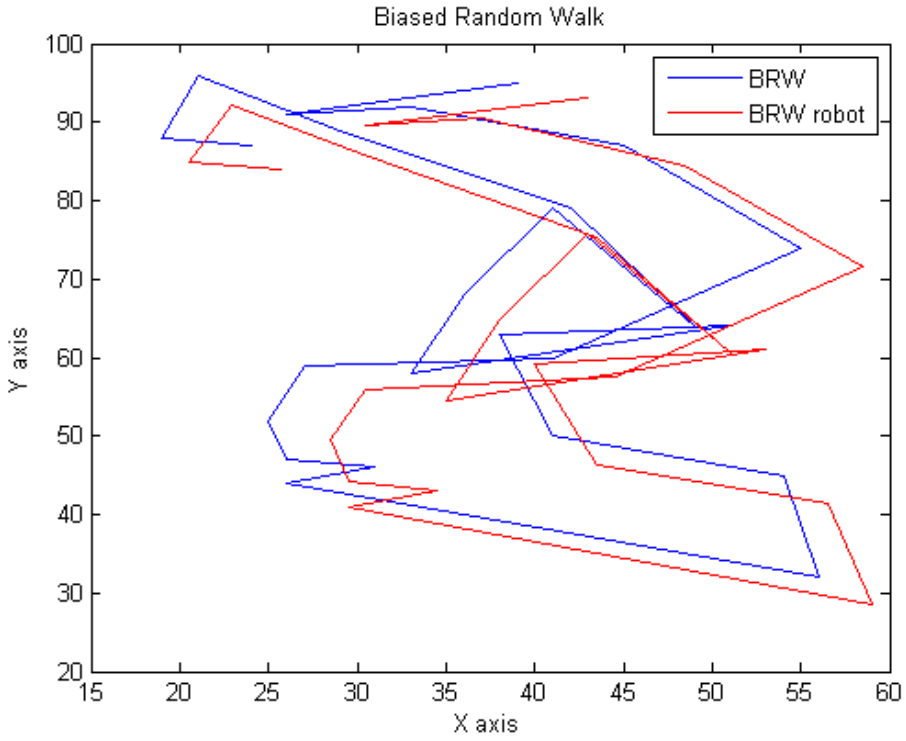

Figure 6: Comparison BRW, between Computer and Robot

First experiment - Conclusion

Looking at (Figure 6), the robot (red line) performs the same path that (blue line), despite the path is shifted slightly from original it keeps same shape. The principal problem detected is when robot executes a motion which involves a long path since the error of position is increased, whilst if it is executing short path, the error of position can be considerate acceptable. "Basic actions", that are "Run" and "Tumble", explained in "Detailed specification", are achieved since each order received in the robot turns the degrees needed to facing objective coordinates and then goes to those coordinates. The error of position is due to deterministic errors in this experiment. Thus, deterministic errors are slight differences in wheels diameter or in motors speed.

Moreover, errors coming from encoders are possible since they have to return an integer value and motors are represented as a continue function really, loosing precision in each reading of each encoder.

Table 4: Comparison coordinates BRW, between Computer and Robot

x BRW	y BRW	x BRW robot	y BRW robot	abs error x	abs error y
39.00	95.00	43.00	93.00	4	2
26.00	91.00	30.50	89.50	4.5	1.5
33.00	92.00	37.00	90.50	4	1.5
45.00	87.00	48.50	84.50	3.5	2.5
55.00	74.00	58.50	71.50	3.5	2.5
41.00	60.00	44.50	57.50	3.5	2.5
27.00	59.00	30.50	56.00	3.5	3.0
25.00	52.00	28.50	49.50	3.5	2.5
26.00	47.00	29.50	44.20	3.5	2.8
31.00	46.00	34.50	43.10	3.5	2.9
26.00	44.00	29.50	41.00	3.5	3.0
56.00	32.00	59.00	28.60	3.0	3.4
54.00	45.00	56.50	41.50	2.5	3.5
41.00	50.00	43.50	46.20	2.5	3.8
38.00	63.00	40.00	59.20	2.0	3.8
51.00	64.00	53.00	61.00	2.0	3.0
33.00	58.00	35.00	54.50	2.0	3.5
36.00	68.00	38.00	64.70	2.0	3.3
41.00	79.00	43.00	75.70	2.0	3.3
49.00	64.00	51.00	60.50	2.0	3.5
42.00	79.00	43.50	75.40	1.5	3.6
29.00	89.00	30.70	85.50	1.7	3.5
21.00	96.00	22.90	92.20	1.9	3.8
19.00	88.00	20.50	85.00	1.5	3.0
24.00	87.00	25.70	84.00	1.7	3.0

Second experiment

For second experiment has been used three e-pucks simultaneously with the aim of achieve every requirement mentioned in "Detailed specification".

Second experiment - Scenario

The scenario for this experiment is:

- *e-puck_2478*, *e-puck_2479* and *e-puck_2481* simultaneously.
- The start coordinates are $(90.00, 60.00)$ for *e-puck_2478*, $(90.00, 50.00)$ for *e-puck_2479* and $(100.00, 55.00)$ for *e-puck_2481*.
- They receive twenty-five orders.
- The speed to move forward the robot is 800 steps.
- The speed to turn the robot is 1000 steps.
- Threshold for proximity are 650, 480, 620 respectively, keeping numeric sort in the e-pucks.

- The environment where the robot is placed is an enclosed surface with green carpet floor.
- Each order is received from computer through Bluetooth.

Second experiment - Results

Results are shown in (Table 5). That summarizes the important things performed for each e-puck. *Start time* means start sequence, being **e-puck_2479** the first. *End time* refers to

Table 5: Second experiment

	e-puck_2478	**e-puck_2479**	**e-puck_2481**
Start time	Third	First	Second
End time	First	N/A	Third
Overall time	61.532501 seconds	N/A	60.659618 seconds
Completed orders	$\frac{25}{25}$	$\frac{7}{25}$	$\frac{24}{25}$
Overcomes threshold-proximity	3	N/A	11
Obstacles avoids	$\frac{1}{3}$	N/A	$\frac{4}{11}$
Ambient light influences	$\frac{2}{3}$	N/A	$\frac{7}{11}$

sequence for finishing twenty-five orders, concluding that **e-puck_2479** is locked at order eight. *Overall time* means elapsed time for performing twenty-five orders, where is not added pause times neither communication times. *Completed orders* refers to number of coordinates that has been reached respect to total. *Overcomes threshold-proximity* means the number of times that sensors detect something. *Obstacles avoids* refers obstacles avoided during the experiment, where **e-puck_2481** detects once a robot **e-puck_2478** and three times detects a wall. *Ambient light influences* means number of times which is detected something, not being this an obstacle.

The output created by Java is shown in (Listing 22) and the video executing this experiment can be downloaded from [18]. For this experiment is need to calibrate proximity sensors. There is a video [16] which shows the execution of calibration procedure which allows to get the value for threshold in each e-puck.

Second experiment - Conclusion

"Basic actions" has been proved in the first experiment but can be highlighted that is need to use another approach to estimate better the position of the robot since the paths which the robot runs can be shorts or long, having to work well in every situation.

"Protocol avoidance" requirements are achieved since there are three robots located at the environment and they are avoiding to crash each other and walls. Despite sensor proximity has been calibrated before, there is noise due to ambient light, although it does not influence for detecting the obstacles.

Regarding "Concurrence" is satisfied since three robots are running simultaneously in one computer. Although, **e-puck_2479** is locked after seven orders it can be due to computer which

runs the program has resources limitations or due to data loss through Bluetooth communication. If it is run this experiment in another computer, over seventy orders without losing communication are reached. Hence, "Concurrence" is achieved also.

Conclusions

BRW is one of the behaviours which performs Escherichia Coli and it is implemented in ARN. ARN approach is a source of inspiration for creating new AI techniques and can be applied for simulating cellular intelligence in robots. Thus, **e-puck** robot has been selected as a candidate for executing BRW, being new in the research group that works with ARN where there has been the need to study characteristics which provide the e-puck.

After the characteristics has been studied, the need to develop a protocol for **communicating** the robot with **Java platform** has been discovered since no API developed in Java exists. Thus, after creating this protocol the understandability between the device and Java platform is possible.

The next step has been the programming of the behaviour associated with BRW which are "Runs" and "Tumbles". This behaviour has been implemented using **Odometry** approach since there is need to know where the robot is located in the environment every-time. Despite this approach does not work with accuracy, for data evaluated in the first experiment, it is however acceptable, taking into account that this data corresponds with real behaviour performed by BRW in ARN. Hence, "Basic Actions" have been achieved, getting good results when robot runs short paths and increasing the error estimation of position when robot runs long paths. But there are errors that can be solved, those are **systemic errors**, so a proper calibration can remove them, improving estimation of position. Some of these systemic errors are (misalignment of the wheels, slight differences in wheels diameter or even slight differences in motors speed). However, there are **non-systemic errors** where removal without use an external source is impossible. Some of these non-systemic errors are (slippage of the wheels, variation in the contact point of the wheel).

Regarding **obstacles avoiding**, this requirement is very important since the scenario of the project involves running three e-puck simultaneously without crashing with the walls. This requirement has been solved using Distal approach, concluding that is working very well if proximity sensors are calibrated correctly. Indeed, to obtain a good calibration is available a method which runs in the environment where the robots must execute BRW, since the value for threshold is changing every-time due to influence of ambient light. However, possibility of differentiate walls or other robots when threshold is overcome does not exist, concluding that the use of the camera which robots have or an external camera, removing uncertainty and improving avoidance protocol is needed. This improvement of avoidance protocol would allow remove *try* approach used currently and could improve the response time for deciding actions to do when something is near, so using another sensor like the camera, the system would have more information over the environment.

Talking about **concurrence**, this requirement is solved through Java platform, allowing every request coming from e-puck simultaneously and establishing the needed resources for each e-puck, concluding that approach selected works well for project purposes. However, as the communication is done via Bluetooth some data can be lost during execution time and is thought that improving communication using errors control, would remove this loss and would avoid robots becoming locked. It should also be noted that it is impossible to manage robots simultaneously in a computer having resources limitations.

Although, e-puck robot has resources limitation it has responded very well for project purposes since BRW behaviour can be executed without any problem, so using this robot for future project related with cellular intelligence could be carried out, but the most important thing now is that it is possible to do complicated calculations in Java side, using a computer with good computational resources and after transferring to e-puck basic actions through Bluetooth, allowing the simulation of other behaviours of cellular intelligence.

Hence, it can be affirmed that e-puck robot is available for performing experiments related with ARN and now the first contact with e-puck has been established, allowing other new projects to focus on other behaviours provided by ARN.

Reflections

In this chapter possible improvement are talked about with the intention of getting better results in the simulation of cellular intelligence.

Firstly, to **improve motions** that robot performs, using odometry is not enough since the removal of systemic errors is first needed and after an estimation of position should be used through external source, that is, the camera available in the lab where robots run experiments. Regarding the use of the camera, work is currently been done to eliminate the distortion that it has and with a software for recognizing the objects located in a picture. This software is imageJ and now is capable of recognize position of robots in the picture, although due to distortion of camera, these positions are wrong.

Secondly, to **differentiate objects** detected by proximity sensors in protocol avoidance, the camera that robot has can be used for sending by Bluetooth the image (when threshold is overcome), so it would allow this uncertainty to be removed doing system more accurate.

Thirdly, at the moment **Bluetooth communication** has been customized for this project, that is, really is like an order interpreter adapted for project requirements. Thus, developing an API in Java which implements a mapping of standard library of the e-puck can be considered useful doing more extendible communication and avoiding programming in two languages simultaneously.

Bibliography

[1] Bluecove 2.1.0 API. Interface discoverylistener. `http://bluecove.org/bluecove/apidocs/index.html`, 2006.

[2] Kok Seng CHONG and Lindsay KLEEMAN. Accurate odometry and error modelling for a mobile robot. *MECSE*, June 1996.

[3] Cyberbotics. Webots overview. `http://www.cyberbotics.com/overview`, 2012.

[4] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, February 2000.

[5] Brian J. Ford. Are cells ingenious? *MICROSCOPE*, 52:3/4:135–144, September 2004.

[6] Claire E. Gerrard, John McCall, George Coghill, and Christopher MacLeod. Artificial reaction networks. *11th UK workshop on Computational Intelligence, University of Manchester*, September 2011.

[7] John McCarthy. What is artificial intelligence? `http://www-formal.stanford.edu/jmc/whatisai/whatisai.html`, November 2007.

[8] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Chistophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. Technical report, École Polytechnique Fédérale de Lausanne, 2009.

[9] University of Colorado. Introduction to robotics, obstacle avoidance. `http://correll.cs.colorado.edu/?p=974`, September 2011.

[10] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. February 2004.

[11] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to autonomous mobile robots*. Bradford Books, 2004.

[12] API specification for the Java 2 Platform. Class properties. `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Properties.html`, 2010.

[13] API specification for the Java 2 Platform. Class thread. `http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html`, 2010.

[14] Robert Gordon University. Code implemented in c side. `https://www.dropbox.com/sh/n4yz3w9o91jss2t/ooeBdCWc4C`, Julio 2012.

[15] Robert Gordon University. Code implemented in java side. `https://www.dropbox.com/sh/90q1zcgnzhinz1y/Z_KjtUitAA`, Julio 2012.

[16] Robert Gordon University. Video for calibrating proximity sensors. https://www.dropbox.com/s/v08ebi5lh2seler/CalibrateSensors.avi, Julio 2012.

[17] Robert Gordon University. Video for first experiment. https://www.dropbox.com/s/i9ufm4jp8cjjjm6/FirstExperiment.avi, Julio 2012.

[18] Robert Gordon University. Video for second experiment. https://www.dropbox.com/s/wmjtgbscduwvfna/SecondExperiment.avi, Julio 2012.

[19] Angel Valera, Marina Valles, Leonardo Marin, and Pedro Albertos. Design and implementation of kalman filters applied to lego nxt based robots. *18th IFAC World Congress*, Agost 2011.

[20] Wikibooks. E-puck and webots. http://en.wikibooks.org/wiki/Cyberbotics%27_Robot_Curriculum/E-puck_and_Webots, March 2011.

[21] École Polytechnique Fédérale de Lausanne. Mechanics of the e-puck. http://www.e-puck.org/index.php?option=com_content&view=article&id=7&Itemid=9, 2010.

[22] École Polytechnique Fédérale de Lausanne. E-puck education robot. http://www.e-puck.org/index.php, 2012.

Program listings

In this chapter, part of the code implemented in the project is attached, there are code for two languages C and Java. The entire Java code can be downloaded from [15] and C code from [14].

Listing 3: C. Atan function customized

```
1 double get_angle_calibrated(double x, double y, double x_, double y_
    ){
2         double angle = get_angle(x,y,x_,y_);
3
4         if(y == y_ && x > x_) return PI_;  /*x axis, left direction
            */
5         else if(x > x_ && y < y_) return (angle + PI_); /*second
            quadrant*/
6         else if(x > x_ && y > y_) return -1*(PI_ - angle); /*third
            quadrant*/
7         else return angle;
8 }
```

Listing 4: C. Distal approach

```
1 int something_near(void){
2         OBSTACLE = 0;
3         if((e_get_prox(0)+e_get_prox(1)) > NEAR) { OBSTACLE = 1;
            OBSTACLES_DETECTED++; return 1; }
4         if((e_get_prox(6)+e_get_prox(7)) > NEAR) { OBSTACLE = 2;
            OBSTACLES_DETECTED++; return 1; }
5         return 0;
6 }
```

Listing 5: C. Odometry forward

```
1 void odometry_forward(){
2         steps_left_now = e_get_steps_left();
3         steps_right_now = e_get_steps_right();
4         left_increment = WHEEL_DIAMETER/10*PI_*(abs(
            STEPS_ELAPSED_LEFT - steps_left_now))/WHEEL_REVOLUTION;
5         right_increment = WHEEL_DIAMETER/10*PI_*(abs(
            STEPS_ELAPSED_RIGHT - steps_right_now))/WHEEL_REVOLUTION;
6         CURRENT_X = CURRENT_X + (right_increment+left_increment)/2*
            cos(LOOK_AT_ROBOT);
7         CURRENT_Y = CURRENT_Y + (right_increment+left_increment)/2*
            sin(LOOK_AT_ROBOT);
```

```
8           STEPS_ELAPSED_LEFT = steps_left_now;
9           STEPS_ELAPSED_RIGHT = steps_right_now;
10          TIME_ELAPSED = TIME_ELAPSED + T;
11  }
```

Listing 6: C. Odometry counterclockwise

```
1   void odometry_counterclockwise(){
2           steps_left_now = e_get_steps_left();
3           steps_right_now = e_get_steps_right();
4           left_increment = WHEEL_DIAMETER*PI_*-1*(abs(
               STEPS_ELAPSED_LEFT - steps_left_now))/WHEEL_REVOLUTION;
5           right_increment = WHEEL_DIAMETER*PI_*(abs(
               STEPS_ELAPSED_RIGHT - steps_right_now))/WHEEL_REVOLUTION;
6           LOOK_AT_ROBOT = LOOK_AT_ROBOT + (right_increment-
               left_increment)/WHEELS_DIAMETER;
7           STEPS_ELAPSED_LEFT = steps_left_now;
8           STEPS_ELAPSED_RIGHT = steps_right_now;
9           TIME_ELAPSED = TIME_ELAPSED + T;
10  }
```

Listing 7: C. Odometry clockwise

```
1   void odometry_clockwise(){
2           steps_left_now = e_get_steps_left();
3           steps_right_now = e_get_steps_right();
4           left_increment = WHEEL_DIAMETER*PI_*(abs(STEPS_ELAPSED_LEFT
               - steps_left_now))/WHEEL_REVOLUTION;
5           right_increment = WHEEL_DIAMETER*PI_*-1*(abs(
               STEPS_ELAPSED_RIGHT - steps_right_now))/WHEEL_REVOLUTION;
6           LOOK_AT_ROBOT = LOOK_AT_ROBOT + (right_increment-
               left_increment)/WHEELS_DIAMETER;
7           STEPS_ELAPSED_LEFT = steps_left_now;
8           STEPS_ELAPSED_RIGHT = steps_right_now;
9           TIME_ELAPSED = TIME_ELAPSED + T;
10  }
```

Listing 8: Java. Methods class

```
1   public class Methods extends Thread{
2           private Epuck e;
3           private Integer m;
4           public Methods(Epuck e,Integer m){
5                   this.e = e;
6                   this.m = m;
7           }
8           public void run() {
9                   super.run();
10                  if(m==1)
11                          this.calibrate_ir_sensors();
12                  if(m==2)
13                          this.runRBW();
```

```
14                  if(m==3)
15                          this.runRBWAvoidingObstacles();
16                  if(m==4)
17                          this.testBluetoothCommunication();
18                  if(m==5)
19                          this.odometry_errors();
20          }
21  }
```

Listing 9: Java. Connect with E-puck by Bluetooth

```
1   public Boolean connect(String bluetoothAddress){
2           try{
3                   sc = (StreamConnection)Connector.open("btspp://"+
                        bluetoothAddress+":1");
4                   is = sc.openInputStream();
5                   os = sc.openOutputStream();
6                   while(is.read()!=3){;}
7                   return true;
8           }
9           catch(IOException ioe){
10                  return false;
11          }
12  }
```

Listing 10: Java. Disconnect with E-puck by Bluetooth

```
1   public Boolean disconnect(){
2           try{
3                   is.close();
4                   os.close();
5                   sc.close();
6                   return true;
7           }
8           catch(IOException ioe){
9                   return false;
10          }
11  }
```

Listing 11: Java. Send message to E-puck by Bluetooth

```
1   public Boolean send(String data){
2           try{
3                   os.write(2);
4                   os.flush();
5                   for(int i=0;i<data.length();i++){
6                           os.write((int)data.charAt(i));
7                           os.flush();
8                   }
9                   os.write(3);
10                  os.flush();
11                  return true;
```

```
12          }
13          catch(IOException ioe){
14                  return false;
15          }
16  }
```

Listing 12: Java. Receive message from E-puck by Bluetooth

```
1   public String receive(){
2           String data="";
3           int d;
4           try{
5                   while((d = is.read())!= 3){
6                           if(d != 2){
7                                   data = data + (char)d;
8                           }
9                   }
10                  return data;
11          }
12          catch(IOException ioe){
13                  return "";
14          }
15  }
```

Listing 13: C. Send message to Java by Bluetooth

```
1   void send(char *output, int length){
2           char b[TAMBUFFER];
3           sprintf(b,"%c%s%c",2,output,3);
4           e_send_uart1_char(b, length+2);
5   }
```

Listing 14: C. Receive message from Java by Bluetooth

```
1   char * receive(int *length){
2           char c;
3           char * data = (char*)malloc(TAMBUFFER);
4           int i = 0;
5           e_set_body_led(1);
6           while(1){
7                   while(!e_ischar_uart1()){;} /* Wait until arrive a
                        message */
8                   while(e_ischar_uart1() !=0) {   /*While the robot
                        receive data, store it*/
9                           e_getchar_uart1(&c);
10                          if(c != 2 && c !=3){
11                                  *(data+i) = (char)c;
12                                  i++;
13                          }
14                  }
15                  if(c == 3){ break; }
16          }
```

```
17          e_set_body_led(0);
18          *length = i;
19          return data;
20  }
```

Listing 15: C. Regular expression for two doubles received

```
1   void regExp(char *data,int length,double *x,double *y){
2           char xs[20], ys[20];
3           int i = 0, j = 0;
4                   i++; /*First character represents initial letter*/
5                   while(*(data+i) != ';'){
6                           xs[j] = *(data+i);
7                           j++;
8                           i++;
9                   }
10                  *x = atof(xs);
11                  i++;
12                  j = 0;
13                  while(i<length){
14                          ys[j] = *(data+i);
15                          j++;
16                          i++;
17                  }
18                  *y = atof(ys);
19  }
```

Listing 16: C. Regular expression for one double received

```
1   void regExp2(char *data,int length,double *x){
2           char xs[20];
3           int i = 0, j = 0;
4                   i++; /*First character represents initial letter*/
5                   while(i<length){
6                           xs[j] = *(data+i);
7                           j++;
8                           i++;
9                   }
10                  *x = atof(xs);
11  }
```

Listing 17: C. Regular expression for one integer received

```
1   void regExp3(char *data,int length,int *n){
2           char xs[20];
3           int i = 0, j = 0;
4                   i++; /*First character represents initial letter*/
5                   while(i<length){
6                           xs[j] = *(data+i);
7                           j++;
8                           i++;
9                   }
```

```
10                  *n = atoi(xs);
11  }
```

Listing 18: C. Orders interpreter

```
1  void protocol(char *data, int length){
2          char output[TAMBUFFER];
3          double x,y;
4          int success;
5          int n; /*Variable to set proximity sensor*/
6                  switch(*(data+0)){
7                          case 'A' :
8                                  regExp2(data,length,&x);
9                                  sprintf(output,"SEND_ANGLE(%f)",x);
10                                 send(output,strlen(output));
11                                 LOOK_AT_ROBOT = x;
12                                 break;
13                         case 'D' :
14                                 sprintf(output,"X:%f,Y:%f,ANGLE:%f",
                                       CURRENT_X,CURRENT_Y,LOOK_AT_ROBOT
                                       );
15                                 send(output,strlen(output));
16                                 break;
17                         case 'G' :
18                                 regExp(data,length,&x,&y);
19                                 go_to(x,y);
20                                 sprintf(output,"GO_TO(%f,%f)",x,y);
21                                 send(output,strlen(output));
22                                 break;
23                         case 'N' :
24                                 regExp3(data,length,&n);
25                                 sprintf(output,"SEND_NEAR(%d)",n);
26                                 send(output,strlen(output));
27                                 NEAR = n;
28                                 break;
29                         case 'O' :
30                                 regExp(data,length,&x,&y);
31                                 success = go_to_avoiding_obstacles(x
                                       ,y);
32                                 if(success == 1){ sprintf(output,"
                                       GO_TO_AVOIDING(%f,%f)",CURRENT_X,
                                       CURRENT_Y); }
33                                 else { sprintf(output,"STOPS_AT(%f,%
                                       f)",CURRENT_X,CURRENT_Y); }
34                                 send(output,strlen(output));
35                                 break;
36                         case 'P' :
37                                 regExp(data,length,&x,&y);
38                                 sprintf(output,"SEND_POSITION(%f,%f)
                                       ",x,y);
39                                 send(output,strlen(output));
```

```
40                          CURRENT_X = x;
41                          CURRENT_Y = y;
42                          break;
43                  case 'S' :
44                          sprintf(output,"SEND_STOP()");
45                          send(output,strlen(output));
46                          break;
47                  case 'T' :
48                          sprintf(output,"OVERALL_TIME(%f)",
                                OVERALL_TIME);
49                          send(output,strlen(output));
50                          break;
51                  case 'U' :
52                          sprintf(output,"OBSTACLES_DETECTED(%
                                d)",OBSTACLES_DETECTED);
53                          send(output,strlen(output));
54                          break;
55                  case 'X' :
56                          sprintf(output,"PROX0:%d-PROX1:%d-
                                PROX7:%d-PROX6:%d",e_get_prox(0),
                                e_get_prox(1),e_get_prox(7),
                                e_get_prox(6));
57                          send(output,strlen(output));
58                          break;
59                  case 'a':
60                          regExp2(data,length,&x);
61                          go_to_angle(x);
62                          sprintf(output,"%d;%d;%d;%d",
                                e_get_prox(0),e_get_prox(1),
                                e_get_prox(7),e_get_prox(6));
63                          send(output,strlen(output));
64                          break;
65                  case 'b':
66                          regExp2(data,length,&x);
67                          turn_angle(x);
68                          sprintf(output,"GO_TO_ANGLE(%f),TIME
                                ELAPSED(%f)",LOOK_AT_ROBOT,
                                TIME_ELAPSED);
69                          send(output,strlen(output));
70                          break;
71                  case 'c':
72                          regExp(data,length,&x,&y);
73                          forward(x,y);
74                          sprintf(output,"FORWARD,TIME ELAPSED
                                (%f)",TIME_ELAPSED);
75                          send(output,strlen(output));
76                          break;
77                  case 'i' :      /*Start connection*/
78                          sprintf(output,"
                                CONNECTION_ESTABLISHED()");
```

```
79                                  send(output,strlen(output));
80                                  break;
81                          default :
82                                  sprintf(output,"UNKNOWN ORDER,
                                       length string is: %d",length);
83                                  send(output,strlen(output));
84                                  break;
85                  }
```

Listing 19: C. Release memory

```
1  void run_brw(){
2          char *data;
3          int length=0;
4          e_init_motors();
5          e_init_ad_scan(ALL_ADC);          /*Proximity*/
6          e_calibrate_ir(); /*Calibrate proximity sensors*/
7      e_start_agendas_processing();
8          while(1){
9                  data = receive(&length);
10                 protocol(data,length);
11                 free(data);      /*Free memory space used by pointer
                       data*/
12         }
13 }
```

Listing 20: C. Management agenda e-puck

```
1          if(enabled3 == 0){ e_activate_agenda(odometry_forward,T
               *10000); enabled3 = 1;}
2          else e_restart_agenda(odometry_forward);
3          e_set_speed_left(MOTOR_SPEED);
4          e_set_speed_right(MOTOR_SPEED);
5
6          /*Actions to perform here*/
7
8          e_set_speed_left(0);
9          e_set_speed_right(0);
10         e_pause_agenda(odometry_forward);
```

Listing 21: Java. Output first experiment

```
0.Looking for e-pucks connected in computer
1.Launch proximity sensors calibration
2.Launch Biased Random Walk
3.Launch Biased Random Walk with obstacle avoidance
4.Test orders customized
5.Odometry errors
3
BlueCove version 2.1.0 on winsock
Connection fails with e-puck_2478
Order received from e-puck_2479 is SEND_POSITION(4.000000,3.500000)
```

```
Order received from e-puck_2479 is SEND_ANGLE(0.000000)
Order received from e-puck_2479 is SEND_NEAR(6000)
Connection fails with e-puck_2481
Order: 0 received from e-puck_2479 is GO_TO_AVOIDING
   (39.000000,95.000000)
Order: 1 received from e-puck_2479 is GO_TO_AVOIDING
   (26.000000,91.000000)
Order: 2 received from e-puck_2479 is GO_TO_AVOIDING
   (33.000000,92.000000)
Order: 3 received from e-puck_2479 is GO_TO_AVOIDING
   (45.000000,87.000000)
Order: 4 received from e-puck_2479 is GO_TO_AVOIDING
   (55.000000,74.000000)
Order: 5 received from e-puck_2479 is GO_TO_AVOIDING
   (41.000000,60.000000)
Order: 6 received from e-puck_2479 is GO_TO_AVOIDING
   (27.000000,59.000000)
Order: 7 received from e-puck_2479 is GO_TO_AVOIDING
   (25.000000,52.000000)
Order: 8 received from e-puck_2479 is GO_TO_AVOIDING
   (26.000000,47.000000)
Order: 9 received from e-puck_2479 is GO_TO_AVOIDING
   (31.000000,46.000000)
Order: 10 received from e-puck_2479 is GO_TO_AVOIDING
   (26.000000,44.000000)
Order: 11 received from e-puck_2479 is GO_TO_AVOIDING
   (56.000000,32.000000)
Order: 12 received from e-puck_2479 is GO_TO_AVOIDING
   (54.000000,45.000000)
Order: 13 received from e-puck_2479 is GO_TO_AVOIDING
   (41.000000,50.000000)
Order: 14 received from e-puck_2479 is GO_TO_AVOIDING
   (38.000000,63.000000)
Order: 15 received from e-puck_2479 is GO_TO_AVOIDING
   (51.000000,64.000000)
Order: 16 received from e-puck_2479 is GO_TO_AVOIDING
   (33.000000,58.000000)
Order: 17 received from e-puck_2479 is GO_TO_AVOIDING
   (36.000000,68.000000)
Order: 18 received from e-puck_2479 is GO_TO_AVOIDING
   (41.000000,79.000000)
Order: 19 received from e-puck_2479 is GO_TO_AVOIDING
   (49.000000,64.000000)
Order: 20 received from e-puck_2479 is GO_TO_AVOIDING
   (42.000000,79.000000)
Order: 21 received from e-puck_2479 is GO_TO_AVOIDING
   (29.000000,89.000000)
Order: 22 received from e-puck_2479 is GO_TO_AVOIDING
   (21.000000,96.000000)
```

```
Order: 23 received from e-puck_2479 is GO_TO_AVOIDING
   (19.000000,88.000000)
Order: 24 received from e-puck_2479 is GO_TO_AVOIDING
   (24.000000,87.000000)
e-puck_2479 OVERALL_TIME(54.914154)
BlueCove stack shutdown completed
```

Listing 22: Java. Output second experiment

```
0.Looking for e-pucks connected in computer
1.Launch proximity sensors calibration
2.Launch Biased Random Walk
3.Launch Biased Random Walk with obstacle avoidance
4.Test orders customized
5.Odometry errors
3
BlueCove version 2.1.0 on winsock
Order received from e-puck_2478 is SEND_POSITION
   (90.000000,60.000000)
Order received from e-puck_2478 is SEND_ANGLE(0.000000)
Order received from e-puck_2478 is SEND_NEAR(650)
Order received from e-puck_2481 is SEND_POSITION
   (100.000000,55.000000)
Order received from e-puck_2481 is SEND_ANGLE(0.000000)
Order received from e-puck_2481 is SEND_NEAR(620)
Order received from e-puck_2479 is SEND_POSITION
   (90.000000,50.000000)
Order received from e-puck_2479 is SEND_ANGLE(0.000000)
Order received from e-puck_2479 is SEND_NEAR(480)
Order: 0 received from e-puck_2479 is GO_TO_AVOIDING
   (39.000000,95.000000)
Order: 1 received from e-puck_2479 is GO_TO_AVOIDING
   (26.000000,91.000000)
Order: 0 received from e-puck_2481 is GO_TO_AVOIDING
   (38.000000,31.000000)
Order: 0 received from e-puck_2478 is GO_TO_AVOIDING
   (184.000000,27.000000)
Order: 2 received from e-puck_2479 is GO_TO_AVOIDING
   (33.000000,92.000000)
Order: 1 received from e-puck_2481 is GO_TO_AVOIDING
   (38.000000,26.000000)
Order: 3 received from e-puck_2479 is GO_TO_AVOIDING
   (45.000000,87.000000)
Order: 4 received from e-puck_2479 is GO_TO_AVOIDING
   (55.000000,74.000000)
Order: 1 received from e-puck_2478 is GO_TO_AVOIDING
   (134.000000,11.070000)
Order: 2 received from e-puck_2481 is GO_TO_AVOIDING
   (24.000000,19.000000)
Order: 2 received from e-puck_2478 is GO_TO_AVOIDING
   (129.000000,9.011000)
```

```
Order: 5 received from e-puck_2479 is GO_TO_AVOIDING
   (41.000000,60.000000)
Order: 3 received from e-puck_2478 is GO_TO_AVOIDING
   (125.000000,12.000000)
Order: 3 received from e-puck_2481 is GO_TO_AVOIDING
   (59.000000,18.000000)
Order: 6 received from e-puck_2479 is GO_TO_AVOIDING
   (27.000000,59.000000)
Order: 4 received from e-puck_2481 is GO_TO_AVOIDING
   (66.000000,21.000000)
Order: 7 received from e-puck_2479 is GO_TO_AVOIDING
   (25.000000,52.000000)
Order: 4 received from e-puck_2478 is GO_TO_AVOIDING
   (127.000000,20.020000)
Order: 5 received from e-puck_2481 is GO_TO_AVOIDING
   (76.000000,22.000000)
Order: 5 received from e-puck_2478 is GO_TO_AVOIDING
   (118.000000,24.000000)
Order: 6 received from e-puck_2481 is GO_TO_AVOIDING
   (84.000000,31.000000)
Order: 7 received from e-puck_2481 is GO_TO_AVOIDING
   (88.000000,15.000000)
Order: 6 received from e-puck_2478 is GO_TO_AVOIDING
   (108.000000,13.039999)
Order: 8 received from e-puck_2481 is GO_TO_AVOIDING
   (97.000000,11.000000)
Order: 7 received from e-puck_2478 is GO_TO_AVOIDING
   (104.000000,22.030001)
Order: 8 received from e-puck_2478 is GO_TO_AVOIDING
   (98.000000,36.020000)
Order: 9 received from e-puck_2481 is GO_TO_AVOIDING
   (88.000000,8.010000)
Order: 9 received from e-puck_2478 is GO_TO_AVOIDING
   (93.000000,18.000000)
Order: 10 received from e-puck_2481 is GO_TO_AVOIDING
   (87.000000,10.000999)
Order: 10 received from e-puck_2478 is GO_TO_AVOIDING
   (101.000000,27.000000)
Order: 11 received from e-puck_2478 is GO_TO_AVOIDING
   (87.000000,18.070000)
Order: 11 received from e-puck_2481 is GO_TO_AVOIDING
   (80.000000,13.000000)
Order: 12 received from e-puck_2478 is GO_TO_AVOIDING
   (93.000000,29.000000)
Order: 12 received from e-puck_2481 is GO_TO_AVOIDING
   (75.000000,7.030000)
Order: 13 received from e-puck_2481 is GO_TO_AVOIDING
   (75.000000,12.003000)
Order: 13 received from e-puck_2478 is GO_TO_AVOIDING
   (93.000000,13.000000)
```

```
Order: 14 received from e-puck_2478 is GO_TO_AVOIDING
   (101.000000,19.000000)
Order: 15 received from e-puck_2478 is GO_TO_AVOIDING
   (96.000000,30.090000)
Order: 16 received from e-puck_2478 is GO_TO_AVOIDING
   (89.000000,45.000000)
Order: 17 received from e-puck_2478 is GO_TO_AVOIDING
   (95.000000,41.000000)
Order: 14 received from e-puck_2481 is STOPS_AT(68.255150,7.976673)
Order: 18 received from e-puck_2478 is GO_TO_AVOIDING
   (94.000000,20.000000)
Order: 15 received from e-puck_2481 is GO_TO_AVOIDING
   (59.000000,35.000000)
Order: 19 received from e-puck_2478 is GO_TO_AVOIDING
   (105.000000,13.000000)
Order: 20 received from e-puck_2478 is GO_TO_AVOIDING
   (95.000000,14.030001)
Order: 16 received from e-puck_2481 is GO_TO_AVOIDING
   (57.000000,49.000000)
Order: 21 received from e-puck_2478 is GO_TO_AVOIDING
   (101.000000,7.040000)
Order: 17 received from e-puck_2481 is GO_TO_AVOIDING
   (73.000000,47.000000)
Order: 22 received from e-puck_2478 is GO_TO_AVOIDING
   (106.000000,12.000000)
Order: 18 received from e-puck_2481 is GO_TO_AVOIDING
   (83.000000,65.000000)
Order: 23 received from e-puck_2478 is GO_TO_AVOIDING
   (100.000000,20.020000)
Order: 19 received from e-puck_2481 is GO_TO_AVOIDING
   (83.000000,71.000000)
Order: 24 received from e-puck_2478 is GO_TO_AVOIDING
   (91.000000,32.000000)
e-puck_2478 OVERALL_TIME(61.532501)
e-puck_2478 OBSTACLES_DETECTED(3)
Order: 20 received from e-puck_2481 is GO_TO_AVOIDING
   (83.000000,65.000000)
Order: 21 received from e-puck_2481 is GO_TO_AVOIDING
   (80.000000,71.000000)
Order: 22 received from e-puck_2481 is GO_TO_AVOIDING
   (78.000000,76.000000)
Order: 23 received from e-puck_2481 is GO_TO_AVOIDING
   (72.000000,77.000000)
Order: 24 received from e-puck_2481 is GO_TO_AVOIDING
   (68.000000,73.000000)
e-puck_2481 OVERALL_TIME(60.659618)
e-puck_2481 OBSTACLES_DETECTED(11)
```