SIMULATION DOCUMENTATION

(Seventh Stable Version, September 06)

1. FEATURES

Simulation features:

- Easy way to create new Stages.
- Possibility to set up the initial positions of the robots or light clicking on them and moving them before pushing the "b" key (for starting the simulation). Also you can rotate a robot, clicking on it with the right mouse button and click on Rotate Right or Rotate Left entry of the context menu.
- Availability for setting several light sources.

Robot features:

- Proximity sensors: Infrared sensors.
- Communication sensors: Infrared sensors. They are also proximity sensors.
- Color Sensors: each robot has a color sensor.
- Touch Sensor: each robot has a touch sensor in its front.
- Light sensors: each robot has a light sensor.
- Collision avoiding.
- Executing based on Plans.

2. STRUCTURE OF THE SIMULATION FILES

- basics directory: it contains the basic classes of the simulation. Do not change any file of this directory!
- demoAutonomousConfig.tz file. It is explained later how to change it.
- demo directory: it contains the files for the user.
    - demoAutonomousController.tz file: it is the main file for executing the simulation. Do not change it!
    - demoAutonomousAgent.tz file: it contains the main methods for controlling the behavior of each robot. You can modify this file or create new ones. It is explained later how to change it.
- model directory : it contains classes for modeling robots and sensors. Do not change any file of this directory!
    - robot directory: it contains the model of different robots.
    - sensor directory: it contains different sensors for robots
- stages directory: it contains files for creating different stages. Each file contains a text-draw which descripts the stage. Vertical walls are written in the even columns of the file, and horizontal walls of the stage are written in the odd columns of the file.

3. RUNNING THE SIMULATION

2.1. Copy the file stMDLePlugin.tz which is in the root directory of the simulation to the directory <breve_directory>/Plugins. (<breve_directory> is the directory in which Breve has been unpacked). It is only necessary to copy this file the first time.

2.2. In the file demo/demoAutonomousController.tz, change the first line with the full path of the directory of the simulation. For example "C:\simulations\breve_simulation" (do not write a “\” at the end). It is only necessary to change it the first time.
2.3. Change the global variables in the file demo/demoAutonomousConfig.tz file in order to set the most useful simulation for a user. You can change these variables several times. You can view inside the file the usage of each variable.
2.4. Play the file demo/demoAutonomousController.tz into Breve.
2.5. Once you can see the stage and the robots, push key “b” for beginning the simulation.

3. MODIFY THE SIMULATION FOR NEW BEHAVIOURS

3.1 How to create a new Stage:

Create a new text file in the directory "stages". Be careful creating the file. The odd columns of the text file are for the vertical walls and the odd columns of the text file are for horizontal walls. Please, see other stage file of the stages directory for knowing how to create a new one. After, modify the variable STAGE_FILENAME in the configuration file (demo/demoAutonomousConfig.tz) file with the name of the new stage file.

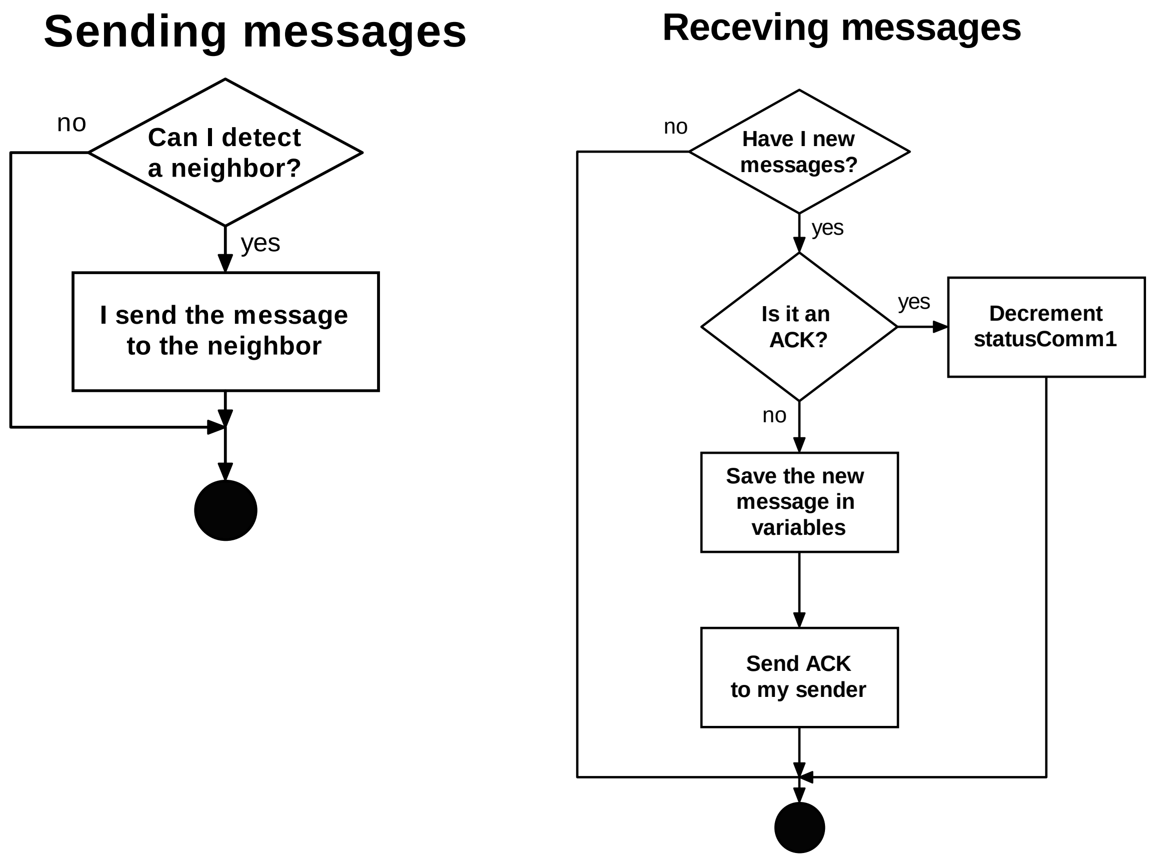
3.2 How to create a new behavior:

You only have to modify the demo/demoAutonomousAgent.tz in order to create new behaviors. In this file you can plan a plan and execute it. It has the same structure than the C files for Jasmine-III robot. The best way to create new behaviors is to copy this file to the same directory (demo directory), rename it and modify it. In this way, you will have the original file with the original behavior (random movement). If you create a copy of the original file, it is necessary to change the line 9 in the file demo/demoAutonomousController.tz file which contains the name of the file (the copy of demo/demoAutonomousAgent.tz). Please change this line with the relative path of your new file.

For creating a new behavior, you must add new plans and also create new behavioral patterns. You can do it in the same way than in the C files. Please read the appendix 1 of this documentation for knowing the available variables and methods. All of these changes are written in the copy of the file demo/demoAutonomousAgent.tz (if you have created a copy of it -recommended-).

4. COMMUNICATION PROTOCOL

The communication Protocol used in the simulation is based on a confirmation protocol. When a robot sends a message to other, the second one sends an acknowledgment message (ACK) to the first one. Only if the ACK is received, the communication has been successful. The following flowchart shows how the protocol works:



5. KNOWN PROBLEMS

- Error including file basics/basicController.tz --> Set correctly the path of your simulation directory. See 2.3 section of this documentation.
- Error including the file “stMDLePlugin.tz” --> Copy correctly the file stMDLePlugin.tz to the <breve_directory>/Plugins. See 2.1 section of this documentation.
- No “Controller” object has been defined --> Please play demo/demoAutonomousContoller.tz in Breve and not a different file.
- Object “obstacle” does not respond to method get-messagesList --> Restart the simulation.
- The stage does not correspond with the "draw" of the text file --> Check if the name of the stage file is set correctly in the variable STAGE_FILENAME of the configuration file (demo/demoAutonomousConfig.tz). If not, check that the text file for the stage is created correctly (vertical walls in even columns and horizontal walls in odd columns). See 3.1 section of this documentation.
- Other problems or suggestions, please contact with me: vprietobrighton@gmail.com

6. APPENDIX 1

6.1 Available variables:

NAME	USAGE
myID(int)	The ID of the robot
statusMain(string)	Contains the main status of the robot. 0 bit- RC flag: 0 - autonomous mode; 1 - manual mode via remote control 1 bit- Obstacle flag: 0 - no obstacles; 1 - obstacle on the way 2 bit- Motion flag: 0 - no current motion; 1 - I'm moving now 3 bit- Direction flag: 0 - motion forwards; 1 - motion backwards 4 bit: Critical collision flag: 0 - there is still open; 1 - no more motion 5 bit: read proximity: 0 - don't read; 1 - read 6 bit: TWI synchronization flag: 0 - slave' last activity not finished; 1 - finished 7 bit: receiving bit: 0 - don't receive(don't listen) anything; 1 - listen for incoming messages
statusComm1(2 ints)	Contains status of send communication. statusComm1[0] represents how many times an output message has to be sent; 64 times max. statusComm1[1] channel that receives the message
statusComm2(string)	Contains status of receiving communication. bits 0-6: channels that should send the message. bit 7: value 0 - no new input messages; Value 1 - there is a new input message.
behavioralRole(int)	Value 0: normal, Value 1: scout, Value 2: leader (you can add more roles with values greater than 2).
localSensors (7 doubles)	Array with sensors data for motion.
localSensorsComm (7 doubles)	Array with sensors data for communications.
receivMessage1(string)	Contains the first word of message obtained via communication.
receivMessage2(string)	Contains the second word of message obtained via communication.
sendMessage1(string)	Contains the first word of message to be sent.
sendMessage2(string)	Contains the second word of message to be sent.
send16(int)	Defines the number of bits of the messages to be sent.

	Value 0: 8 bits; Value 1: 12 bits; Value 2: 16 bits.
receive16(int)	Defines the number of bits of the messages to be received. Value 0: 8 bits; Value 1: 12 bits; Value 2: 16 bits.
statusPlan(int)	Variable for executePlan method.
statusPlanLast(int)	Variable for storing the lastPlan executed.
avoiding(int)	This contains the rotation angle (in degrees) for collision avoiding: small - fine avoiding; large – random motion.
S1Value(double).	Contains the value detected by the left-light sensor. If the sensor does not detect light, this variable values -1. Each robot detects lights from all light sources. The resulting received light is the sum of all received lights from all sources.
S2Value(double).	Contains the value detected by the right-light sensor. If the sensor does not detect light, this variable values -1. Each robot detects lights from all light sources. The resulting received light is the sum of all received lights from all sources.
touchValue(int)	It contains 1 if the touch sensor is active and 0 if the touch sensor is not active.
ColorValue(vector)	It contains the color of the detected object in RGB format. If no object is detected, it values (-1,-1,-1).
energyValue(double)	Contains the value of the available energy of the robot. Initial value=200.
delayActive(int)	Contains the status of the Delay. Value 1: delay is working; Value 0: delay has finished.
TimeoutReached (int)	Contains the status of the Timer. Value 1: time set in the TimerStart has been reached; Value 0: time set in the TimerStart has not been reached yet.

Note: In Breve, byte type does not exist, so we use string type. For access a bit in a register, which is a string, we must use:

valueBit=stringName{bit}.

6.2 Available methods:

In Breve, the way to call methods is the following:

Object nameMethod nameParameter valueParameter.

If the method takes arguments, each argument is associated with a keyword: the keyword identifies which argument will follow. For example, in Breve the method “Read_sensor channel IrSensorId (int)” of the object robot should be called:

Robot Read_sensor channel 2.

All execute lines in Breve must be ended with a dot. Read the Breve Documentation for more details:

http://www.spiderland.org/breve/breve_docs/

The available methods for the simulation are:

NAME	FUNCTIONALITY
Read_sensor channel IrSensorId(int)	Reads the value of the sensor IrSensorId and returns the value read from the ADC. See appendix 2 section in this documentation.
stopRobot	Stops the robot and updates the registers.
moveRobot distance d (int)	Moves the robot an integer distance. If distance==0, it moves continuously.
SetVelocity velocity v(int)	Set the velocity for the robot. “v” can be 1,2,3 or 4. Value 1: highest velocity forward. Value2: middle velocity forwards. Value 3: lowest forwards velocity. Value 4: for backwards at lowest velocity. At the beginning, the initial velocity is set to 1, it means, the maximum value for moving forwards.
Rotate direction d(int) degrees grades (int)	Rotates the robot an angle. Robot turns a small angle each simulation iteration step. If you need to rotate the robot a big angle (more than 10 degrees), it needs some iterations for turning.
RotateFix direction d(int) degrees grades (int)	Rotates the robot an angle. Robot turns the degrees in one iteration step. This method should be used only for setting the initial position of the robot, but for normal turning, use Rotate method.
ReadProximity	Detects if any obstacle is in the way and how far it is.
Read_distance channel IrSensorId(int)	It calls to Read_sensor method. It returns the value read from the ADC. See appendix 2 section in this documentation.
sendACK channel IRChannelID(int)	Sends an ACK to a neighbor ONLY if the robot can see other robot.
send_8bits message msg(string) channel IRChannelID(int)	Sends a message with 8 bits by a specified channel.
send_16bits message1 msg1(string) message2 msg2(string) channel	Sends a message with 12 bits by a specified channel.

IRChannelID(int)	
ReceiveMessages	Detects if someone has sent any message to the robot.
SendMessages	Sends messages with the data saved in sendMessage variables.
resetComm	Resets Communications (disable communications).
GetTouch Value	It gets the value of the touch sensor and save it in touchValue variable. Once the method is called, next statusPlan will be 10.
GetColorValue	It gets the value of the color sensor and save it in ColorValue variable. Once the method is called, next statusPlan will be 16.
GetS1Value	It gets the value of the left-light sensor and save it in S1Value variable. Once the method is called, next statusPlan will be 13.
GetS2Value	It gets the value of the right-light sensor and save it in S2Value variable. Once the method is called, next statusPlan will be 15.
GetEnergyValue	It gets the value of the robot's energy and save it in energyValue variable. Once the method is called, next statusPlan will be 11.
SendMessage8 message message(string) numberSend numberSend(int) channels channels(string)	Prepares the sendMessage variables for sending messages.
Start	This method is called when the user presses the key "b" at the beginning of the simulation. In it, the user can set up the initial variables values for each robot. Also, the user can define different values for the variables depending on the behavior of the robot. This method is set in the demo/demoAutonomousAgent.tz file.
TimerStart iterations i(double)	Set a Timer for "i" simulation iterations. When the Timer is reached, the variable TimeoutReached values 1 and when the timeout is not reached the variable TimeoutReached values 0. Also, when the timer is reached, the execution goes to software interruptions (statusPlan=14).
TimerStop	Stop the Timer active when TimerStart method was called.
Delay seconds s(double)	Provokes a delay of "s" simulation seconds (iteration steps) in the robot. During the delay is working, the robot is stopped and the variable delayActive is set to 1. When the delay has finished, delayActive is clear to 0 and the simulation execution returns to the main cycle. Be careful because the

	execution does not return to the next instruction after the delay called!!!. The statusPlan variable contains the last statusPlan executed.
LED_playB	Set the robot color to blue.
LED_playG	Set the robot color to green.
LED_playR	Set the robot color to red.
LED_play_otherColor color colorVector(vector)	Set the robot color to any color. ColorVector contains a vector with the desired color. The color is represented as a vector RGB. It means (1,0,0) is red color, (0,1,0) is green color, (0,0,1) is blue color.

6.3 Remote Control

You can control a robot using the keyboard. To enter in “remote control mode”, you need to press key “r”. Then, the robot will stop. For controlling it, you can use the keys “i” for moving forwards, key “k” for stopping the robot, key ”m” for moving backwards, key “j” for turning left the robot and key “l” for turning right the robot. While the robot is moving forwards, all the sensors are active, so if the robot detects an obstacle, it will enter into collision avoiding mode. For returning from “remote control mode” to “autonomous cycle mode”, you must press key “a”.

Only one robot can be controlled with the keyboard.

7. APPENDIX 2

7.1 Comparative table between ADC values and real distance in centimeters.

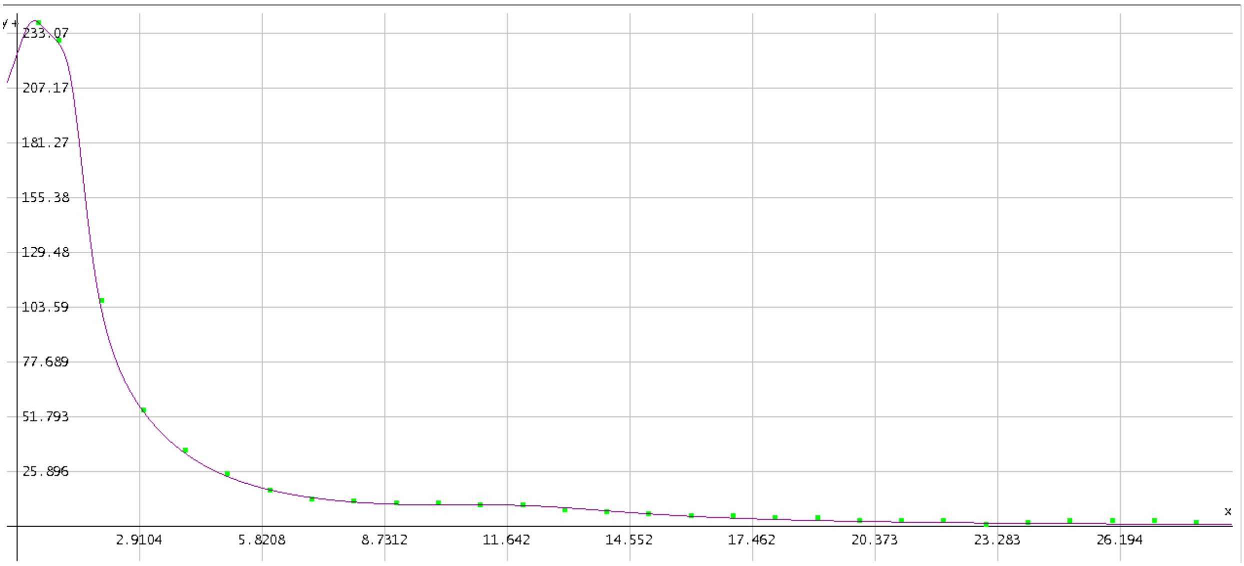
7.1.1 For proximity sensors (Infrared Sensors with 60º aperture angle)

Distance (cm)	ADC Value
0,5	238
1	228
2	103
4	34
6	17
8	11
10	10
12	10
14	7
16	5
18	3
20	2
22	2
24	1

The function applied for obtaining the table of proximity sensors values is:

$$y = \frac{100}{0.3\cdot(x+1.1)^6+1} + \frac{125}{2\cdot(x-0.8)^4+1} + \frac{100}{0.2\cdot(x-0.8)^2+1} + \frac{6}{0.08\cdot(x-12)^2+1}$$

Following is its representation.



Y axis = ADC Value
X axis = Distance (from sensor to obstacle)

Dots represent the real values, and the curve represents the approximate function used.

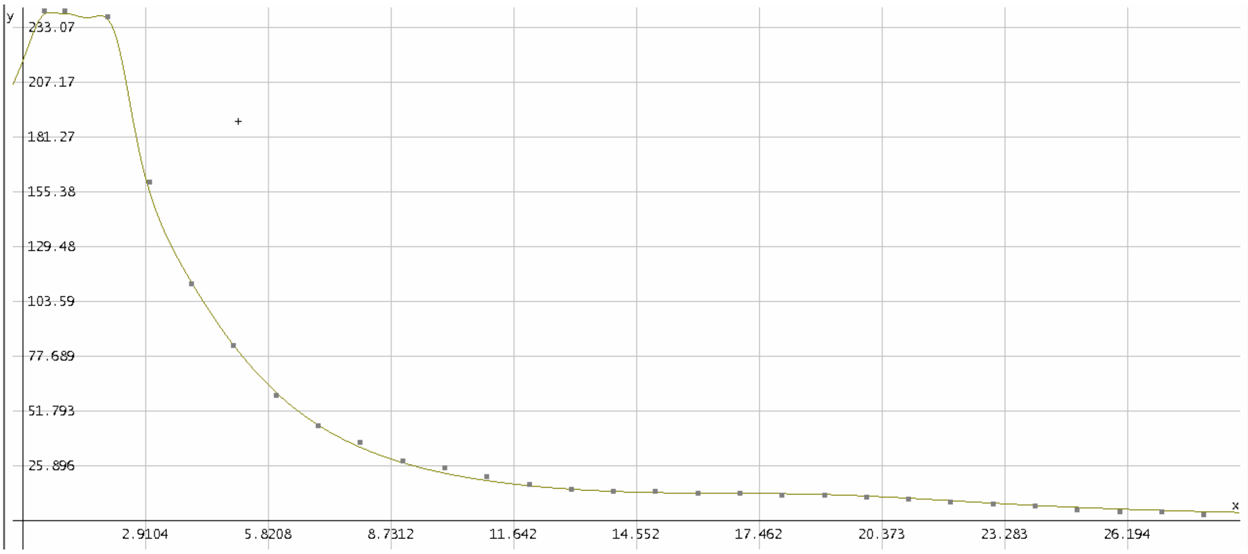
7.1.2 For beamer sensor (Infrared sensor with 15º aperture angle)

Distance (cm)	ADC Value
0,5	246
1	240
2	237
4	113
6	61
8	35
10	22
12	16
14	14
16	13
18	13
20	12
22	9
24	7
26	5
28	4

The function applied for obtaining the table of beamer sensor values is:

$$y = \frac{112}{0.22\cdot(x+0.5)^6+1} + \frac{98}{0.95\cdot(x-1.6)^4+1} + \frac{140}{0.1\cdot(x-2.3)^2+1} + \frac{7.5}{0.03\cdot(x-19)^2+1}$$

Following is its representation:



Y axis = ADC Value
X axis = Distance (from sensor to obstacle)

Dots represent the real values, and the curve represents the approximate function used.