ISSN 2176-5480

22nd International Congress of Mechanical Engineering (COBEM 2013)
November 3-7, 2013, Ribeirão Preto, SP, Brazil
Copyright © 2013 by ABCM

DATA COMMUNICATION AND PROTOCOLS IN SELF-RECONFIGURABLE MODULAR ROBOTS

Ana Carolina Cardoso de Sousa
Dianne Magalhães Viana
Departamento de Engenharia Mecânica, FT, Universidade de Brasília, Asa Norte, CEP 70910-900.
anacsousa1@gmail.com, diannemv@unb.br

Carla Maria Chagas e Cavalcante Koike
Departamento de Ciência da Computação, CIC/EST, Universidade de Brasília, Asa Norte, CEP 70910-900.
ckoike@unb.br

Abstract. *Natural catastrophes, like mudslide and earthquakes are a challenge to the use of high-tech equipments when obstacles block rescue robots with a fixed configuration. Thus, these procedures require more versatile systems. Self-reconfigurable modular robots are autonomous machines with a variable morphology. In addition to the characteristics of conventional robots (such as actuators, sensors and control systems), the modular robots are also able to change its own shape, reorganizing the connectivity of its parts. So, a modular robot has the ability to travel over a wider variety of terrain, adapting well to many tasks. The aim of this project is to develop the structure of communication for modular robots, from hardware to protocols. For tests, we used the ErekoBot, a self-reconfigurable modular robot developed by the Ereko Group, from University of Brasilia (Brazil). After the evaluation of the resources available in the robot, we defined the layer's functions and characteristics of the protocols. Then, we tested both the design of hardware and protocols in real tests in controlled environments. The results of these tests proved the efficiency of the protocol and will offer new support for group projects, specially related to locomotion. The promise of the versatility of modular robotics could lead to a radical change in automation, especially in applications in remote locations where rescue teams cannot reach with traditional equipment.*

Keywords: *Modular Robotics, Data Communication, Protocols.*

1. INTRODUCTION

In situations such as a building collapse due to a landslide or an earthquake, there is occasionally a lack of access by the rescue teams. Thus, some mobile robots were designed to perform this kind of urban operation, like the rescue robot Urbie[1]. A rescue robot gathers information of the ground, processes this data and gets around, dodging obstacles when necessary – all according to predefined rules in its software. But as these disasters are unpredictable, it's impossible to predefine all of them (including physical obstacles) and put it all in the software. So, the usage of robots with fixed configuration is not good choice, as it's known that in certain situations, like when it's unable to move into a passage narrower than its own width, it won't be able to do what it has to.

Research groups study the design and construction of robots with a variable morphology for two purposes: (1) to increase the number of lands that robots can act in and (2) to increase the number of tasks they can perform. In 1994, Yim (Yim (1994)) created the definition of a self -reconfigurable modular robot, a robot that changes its own shape. In the situation where a narrow path is too small for a robot with fixed morphology, a modular robot is able to become smaller and pass through. But, as a new area, there are challenges that difficult mass production of these robots, like manufacturing process, hardware design, control algorithms and communication (Yim *et al.* (2007)). This last one defines how the robot communicates with each of its parts and how each one of its parts communicates with the central computer (host).

The aim of this project is to develop all the structure of the communication for modular robots, since hardware to protocols. For tests, we used the ErekoBot (Fig. 1), the modular robot designed by the Ereko Group of University of Brasilia (Brazil) (Souza (2011)). We defined the layer's characteristics and functions of the model OSI, based on the project's methodology called bottom-up. This type of methodology starts from the definition of the most basic components and then, gradually, increases the complexity to the system.

2. METHODS

2.1 State of the Art

Currently, a number of related approaches exists for programming modular robots. The most basic form is a centralized control application executed on a workstation that remotely controls the modular robot as has been implemented in the early works of MTRAN (Yim *et al.* (2009)) and PolyBot (Zhang *et al.* (2004)). With sensors, it is necessary to have an hybrid system with a global and a local control (Shen *et al.* (2002)): the global actions such as locomotion require a

[1] http://technology.nasa.gov/data/TechFinder/attachments/20060022037_2006145860.pdf

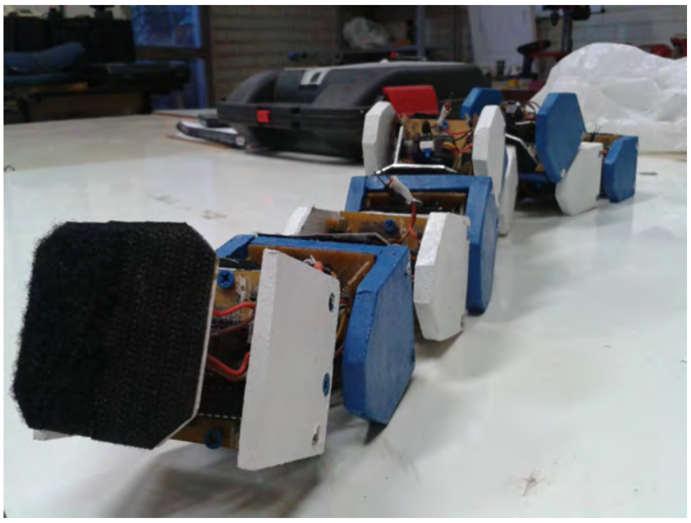


Figure 1. ErekoBots.

re-computation of the local actions to be executed by the individual modules. In most systems, the modules are physically connected and form one single, rigid body with actuators, which have to be coordinated collectively to move the whole robotic organism (Hamann *et al.* (2010)).

These systems, together with the latter developments with ErekoBot $\alpha\, v.2$ prototype were the stating points in developing the Protocol.

2.1.1 ErekoBot $\alpha\, v.2$

The ErekoBot $\alpha\, v.2$ is the latter functional prototype developed by graduate students in Ereko Group at the University of Brasília (UnB). This prototype was designed to be used as a tool to study important aspects of modular systems, such as assembly of different configurations, algorithms for motion and reconfiguration. The module has a similar shape to the MTRAN and PolyBot's modules. Its dimensions are 50mm x 50mm x 50mm and use manual fitting nuts and bolts in the joints of the two sides of the connection. Table 1 resumes the module's characteristics.

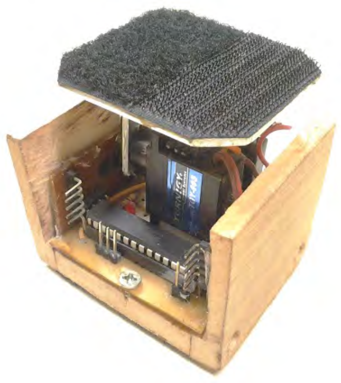


Figure 2. ErekoBot $\alpha\, v.2$.

2.2 Protocol

There are two protocol development methodologies: top-down and bottom-up. The first starts from a broad overview of the proposed system and gradually defines smaller components. The second defines the most basic components and uses them to increase the system's complexity (Tanenbaum (2002)). In this project, we used the methodology bottom-up since the ErekoBot hardware system is known in details. As this is an application in the area of automation, we used a stack 1-2-7 (Physical, Data Link and Application Layers).

ISSN 2176-5480

Table 1. ErekoBot $\alpha\, v.2$'s characteristics

Type	Homogeneous
Dimensions (mm)	50x50x50
Armor material	Wood (eucalyptus)
Architecture	Mobile
Reconfiguration	Deterministic
Self-Reconfiguration	1 (180 rotational degrees)
Pluggable sides	Yes/Manual with Velcro
Degrees of Freedom	2
Servo Motors	Turnigy TGY90S

2.2.1 Physical Layer

Table 2 shows the Physical Layer's characteristics of the Protocol[2].

Table 2. The Physical Layer's Characteristics.

Characteristic	Description
Half-Duplex	The modules and the host can send and receive data, but never simultaneously.
Serial	The transmitter sends the data bit by bit through cables. (The microprocessor Atmel® ATmega 8® uses the communication port USART.)
Asynchronous	The host can send data to the system at any time.
Baud Rate 9600 bps	Every second, Atmel® ATmega 8®sends 9600 bits by the cables.
Interrupt at a low level	Atmel® ATmega 8®'s interruption is low level-triggered.

The physical interface RS-232 (Dept. (1969)) performs the communication between the modules and the computer. Figure 3 shows the interface circuit (the component MAX232 converts the RS-232 and the microcontroller's voltages).

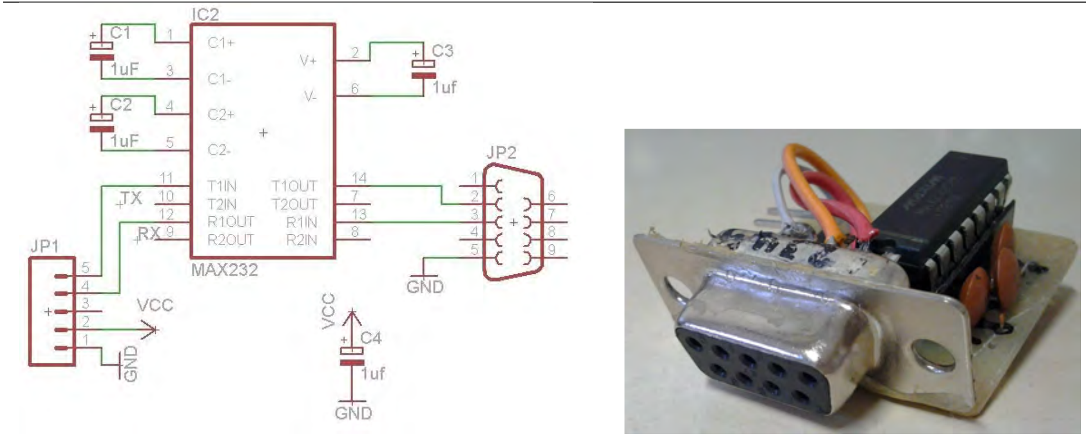


Figure 3. RS-232 interface circuit.

2.2.2 Data Link Layer

Logic Synchronization between entities As Atmel® ATmega 8® synchronizes only receiving bytes, the protocol must ensure the packet synchronization (Sync Logic). Each packet has a single byte reserved value (10H) and when a communication interrupt occurs in a module, it expects approval of that specific byte order to receive the rest of the packet.

[2]They were based in the baseband regular pulse with cable communication (Tanenbaum (2002)).

Packets' Assembly and Delimitation Figure 4 shows two types of protocol packets: the Standard Packet (five bytes) and the Configuration Packet (eleven bytes).

Standard Packet			
Synchronization (1 byte)	Identification (1 byte)	Message (2 bytes)	Checksum (1 byte)

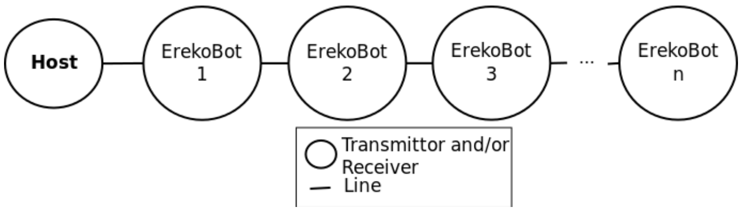
Configuration Packet						
Synchronization (1 byte)	Identification (1 byte)	New Identification (1 byte)	Number of Modules (1 byte)	Period (2 bytes)	Position of the period (1 byte)	Checksum (1 byte)

Figure 4. Packet Types.

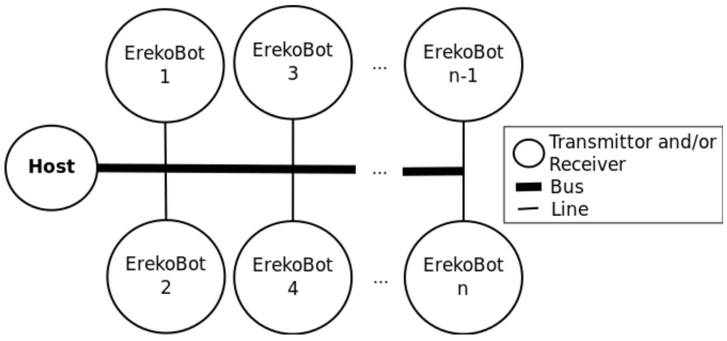
Control of transmission errors At the beginning, each packet has a synchronization byte (sync) and an identification byte (address). And, at the end, there is a byte for error detection (checksum).

Routing The Atmel® ATmega 8® allows only one transmition channel and one reception in each module. Therefore, the topology of the protocol is the type line or bus (Fig. 5). The host transmits the packets to the first ErekoBot, which uses the information addressed to it and passes the rest to the next module.

The topology may also be like a ring, when the last ErekoBot sends messages to the host.



(a)



(b)
Figure 5. Protocol's Topologies (a) Line (b) Bus.

Mapping logical addresses physical All the packets are addressed to a single destination, except for the Start and Stop's Packets.

Upon receiving a packet, each module checks whether the identification field matches its own address. If so, the module uses the message, if not, forwards the packet to the next module (unicast). In the special case of the Start and Stop's Packets, a specific reserved value fills the identification field. When a module receives such a packet, it uses the information and transmits to the others (broadcast).

2.2.3 Application Layer

The two Protocol Packets transmit different types of messages.

The Standard Packet transmits:

- The movement phase;
- The angle position of the servo;
- When the modules start to move;
- When the modules stop to move;
- The transmission of the Configuration Packet.

The Configuration Packet configures ErekoBot motion, stating:

- The new address;
- The Gait Table to be used;
- The number of the modules in the system;
- The period;
- The movement phase (same of the Standard Packet).

There are four types of application in the Protocol, called Communication Profiles[3].

First Profile The First Profile is the simplest, in which the modules do not have any autonomy, the host computes all the values for each module and transmits for the system in real time. This profile only uses the Standard Packet.

The receiver sets up the communication and the PWM signal, and then waits for a communication interruption. In this interruption, the module receives the packet and checks if the identification field is equal to its own address, if so, apply this value in the motor, if not, it sends the packet to the next module (Fig. 6).

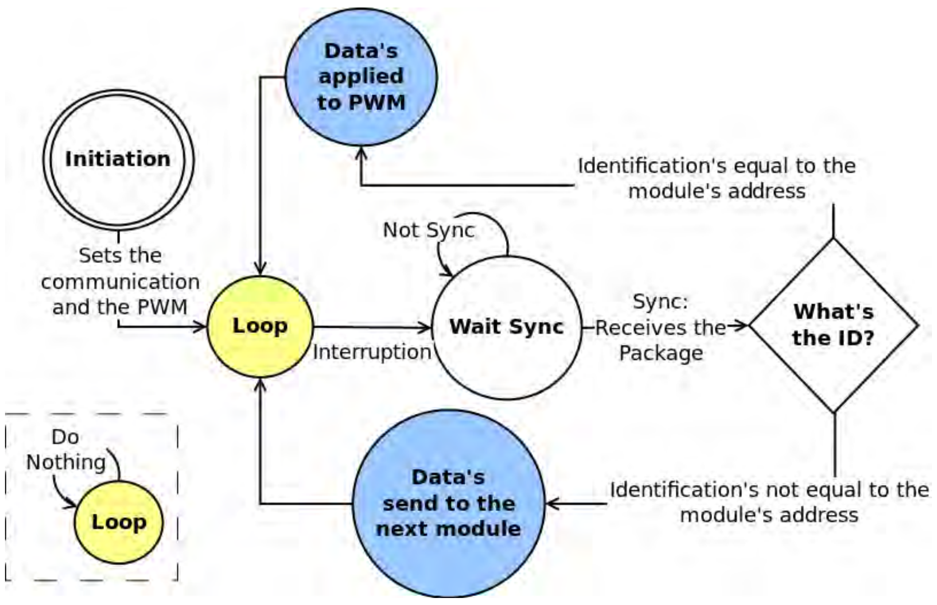

Data's applied to PWM
Initiation
Identification's equal to the module's address
Not Sync
Sets the communication and the PWM
Loop
Wait Sync
Sync: Receives the Package
What's the ID?
Interruption
Do Nothing
Loop
Data's send to the next module
Identification's not equal to the module's address

Figure 6. Receiver's algorithm at the First Profile: unicast (blue) and loop (yellow).

Figure 7 shows the Java program that transmits the packets through the RS-232 interface. In this program, the user determines the number of modules in the system and the period size. From that, we initialize and fill tables with the angles and address values for each module, and then we can start or stop the movement with just one button.

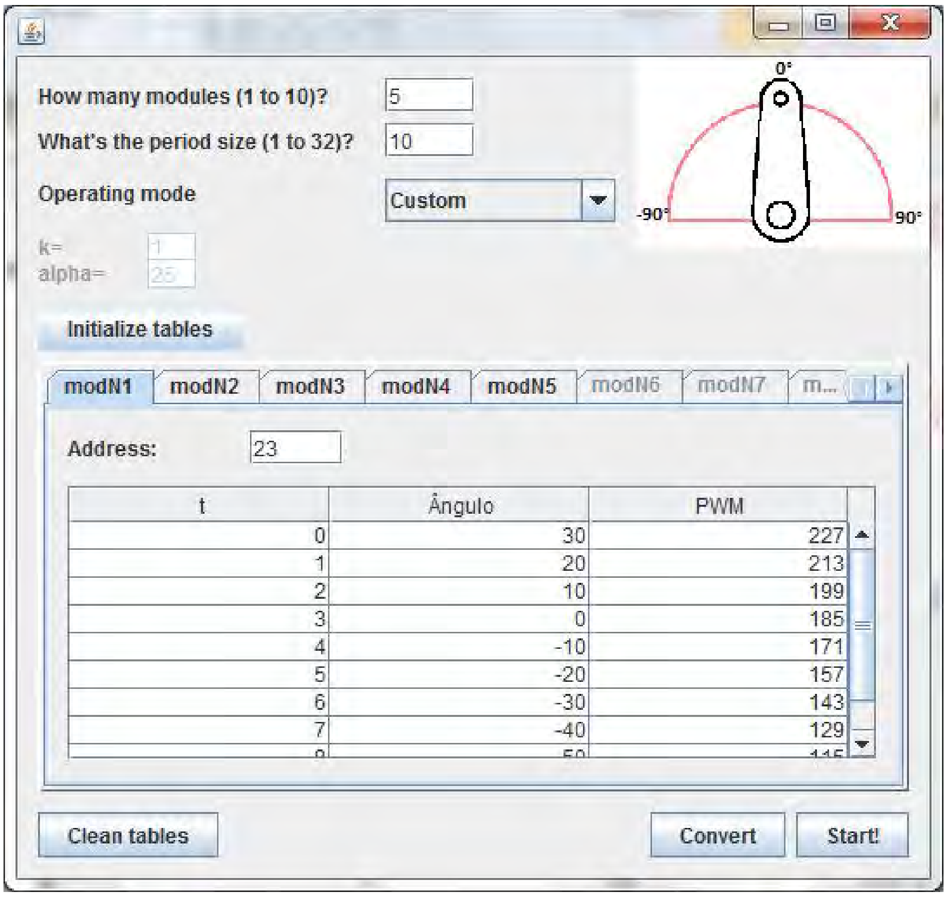
Second Profile In the Second Profile, modules have a Table of Gaits and the host tells the time in which the modules must start or stop the movement. This profile just uses the Standard Packet.

The receiver sets up the communication and pwm signal, and waits for a start signal. With this signal, it searches at the Table of Gaits the value of the servo PWM corresponding to the current time state and applies it at the servo motor.

In the case of a communication interruption, the program receives the packet and checks if it is a broadcast (Start or Stop) or if it is unicast (updates the time current state). Figure 8 shows the receiver algorithm.

[3]All codes are open source and can be downloaded at: http://sourceforge.net/projects/erekobotcommunication/files/?source=navbar

ISSN 2176-5480


How many modules (1 to 10)? 5
What's the period size (1 to 32)? 10
Operating mode: Custom
k= 1
alpha= 25

Address: 23

t	Ângulo	PWM
0	30	227
1	20	213
2	10	199
3	0	185
4	-10	171
5	-20	157
6	-30	143
7	-40	129

Figure 7. Transmitter's program at the First Profile.

Figure 9 shows the Java program that transmits the packets through the RS-232 interface. This program operates similarly to the First Profile: the user determines the system information, initializes and fills the tables and finally sets, starts and stops the motion with the respective buttons.

Third Profile In the Third Profile, the modules have the ability to reconfigure themselves, because they contain more than one Table of Gaits. The host sets up when and which tables will be used, and, for this, it uses the two packets: Standard and Configuration Packets.

The receiver sets up the communication and the pwm signal and waits for a start signal. With this signal, it is searched in the Table of Gaits that has configured the corresponding angle value to the current time and applies it.

In the case of a communication interruption, the program receives the packet and checks if it is broadcast (start or stop) or unicast (sets up the movement). Figure 10 shows the receiver algorithm

Figure 11 shows the Java program that transmits the packets through the RS-232 interface. This program works similarly to the previous profiles.

Fourth Profile In the Forth Profile, the embedded circuits in the modules are reprogrammable, the host can send real-time algorithms, formulas and / or Tables of Gaits.

This profile has not been implemented in this protocol by impracticability hardware.

2.3 Tests

We tested the protocol and we identified problems related to noise and loss of synchronism. After solving these problems, the Reliability Packet Delivery Test and the Estimation of Transmission Delay Test were conducted to prove the protocol usability and efficiency.

2.3.1 Noise

The noise enables the communication interruption and the module goes into a loop waiting for a synchronization signal, which prevented the correct system execution. To avoid this problem, we added a pull-up resistor to each module.

ISSN 2176-5480

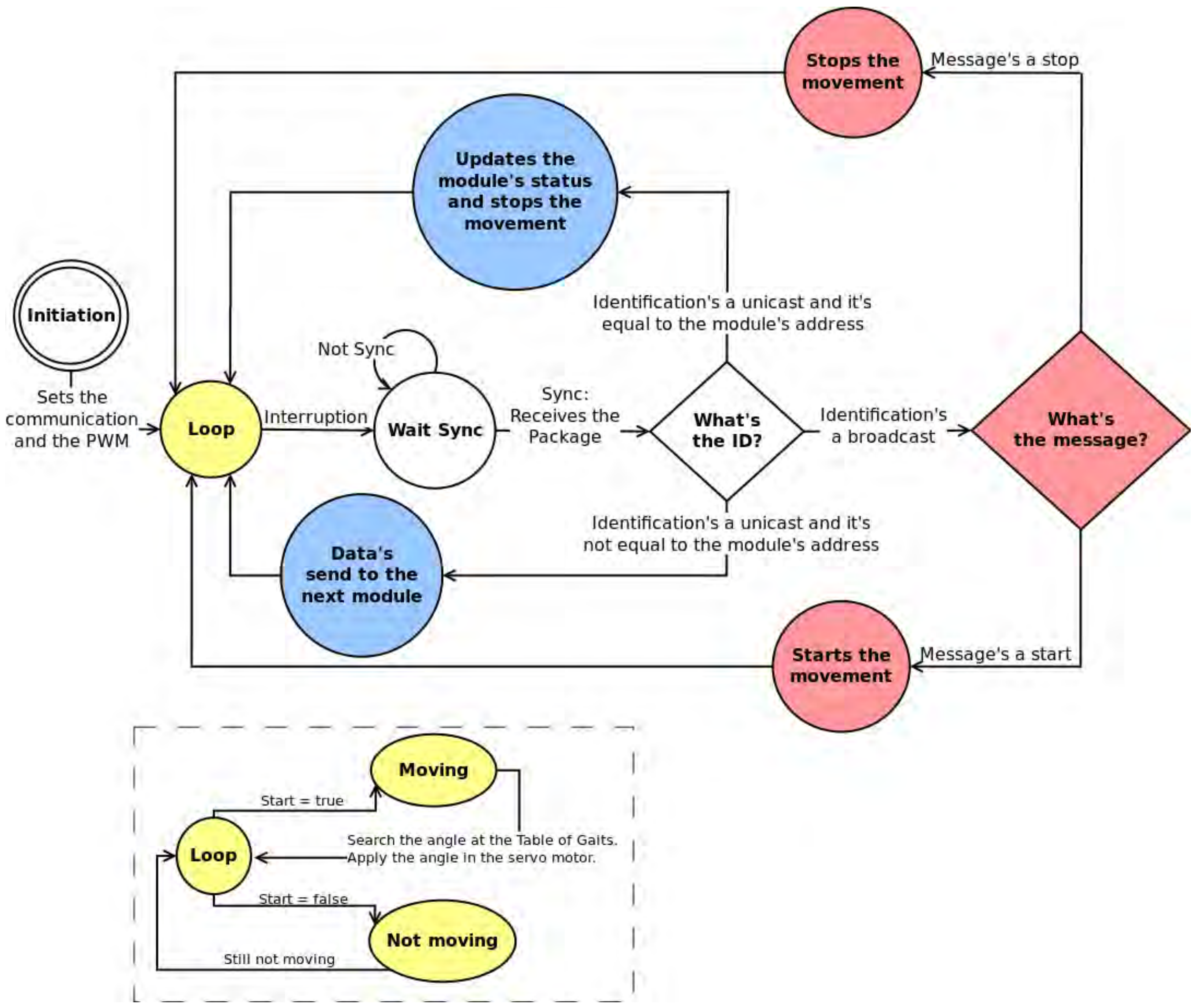
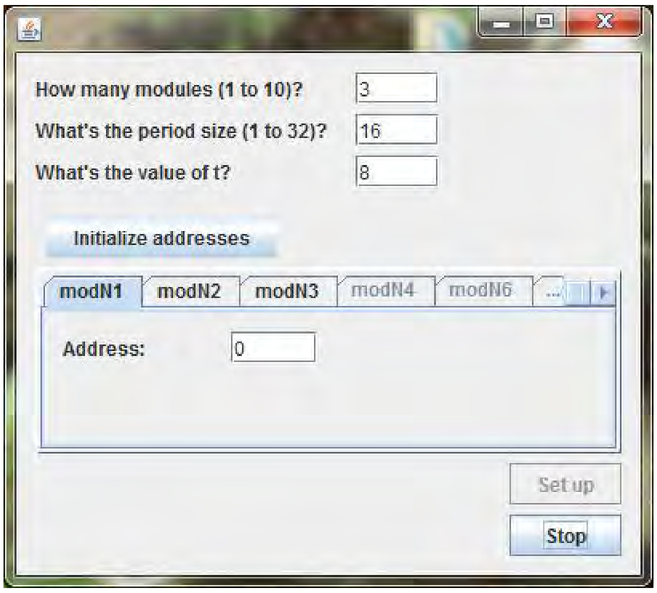

Initiation
Sets the communication and the PWM
Loop
Interruption
Not Sync
Wait Sync
Sync: Receives the Package
What's the ID?
Identification's a unicast and it's equal to the module's address
Updates the module's status and stops the movement
Identification's a unicast and it's not equal to the module's address
Data's send to the next module
Identification's a broadcast
What's the message?
Message's a stop
Stops the movement
Message's a start
Starts the movement
Moving
Start = true
Loop
Search the angle at the Table of Gaits. Apply the angle in the servo motor.
Start = false
Not moving
Still not moving

Figure 8. Receiver's algorithm at the Second Profile: unicast (blue), broadcast (red) and loop (yellow).


How many modules (1 to 10)? 3
What's the period size (1 to 32)? 16
What's the value of t? 8

Address: 0

Figure 9. Transmitter's program at the Second Profile.

To avoid deadlock, we also added a timeout in the Second and Third Profiles: if synchronization does not occur, the module exits the communication thread and returns to the main program.

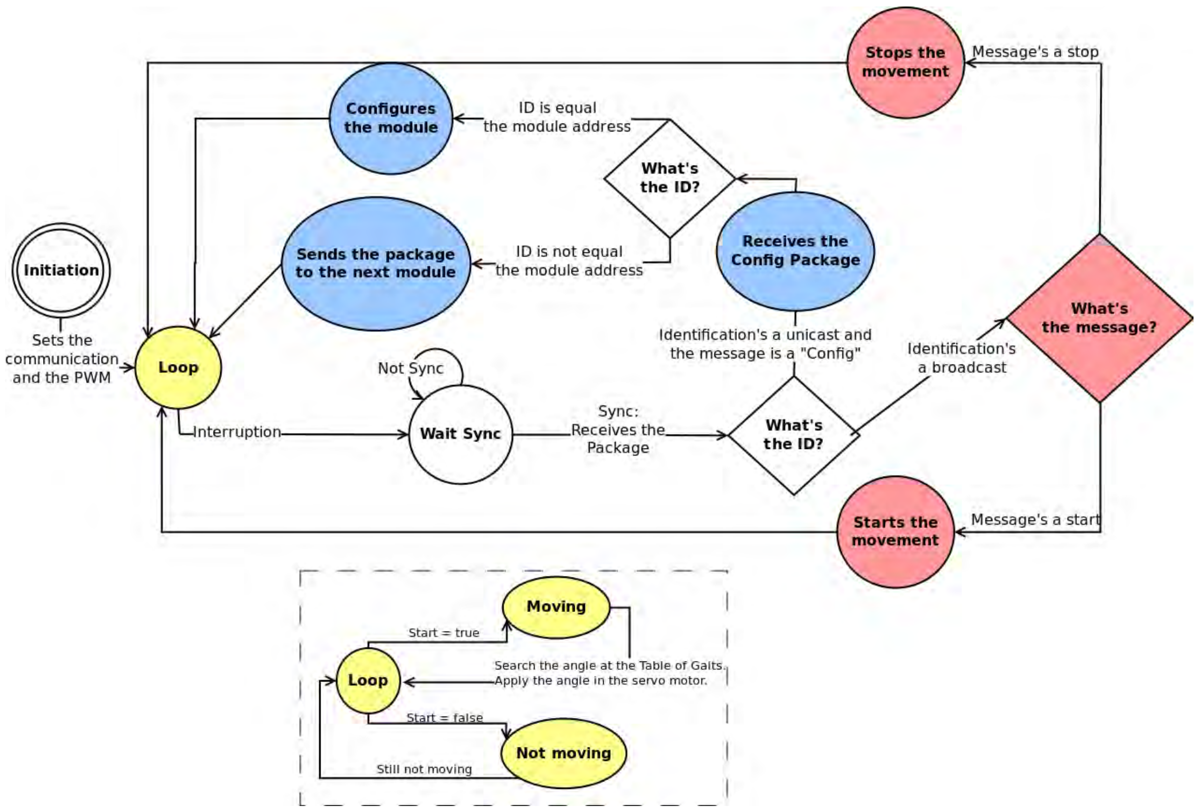
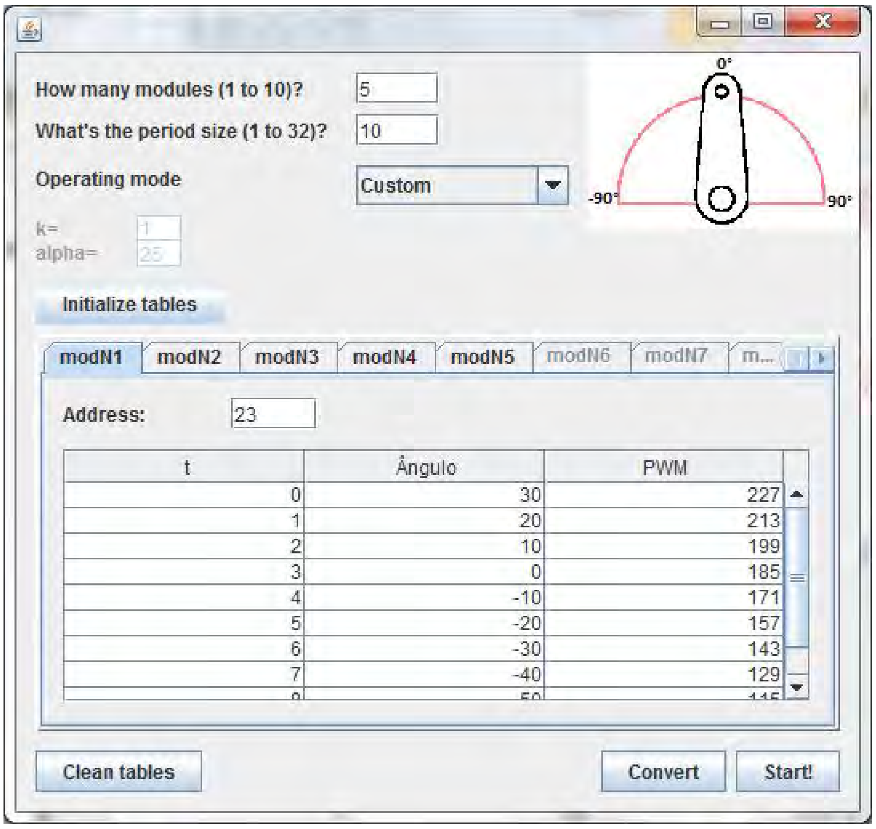

Stops the movement
Message's a stop
Configures the module
ID is equal the module address
What's the ID?
Receives the Config Package
Initiation
Sends the package to the next module
ID is not equal the module address
What's the message?
Identification's a unicast and the message is a "Config"
Identification's a broadcast
Sets the communication and the PWM
Loop
Not Sync
Wait Sync
Sync: Receives the Package
What's the ID?
Interruption
Starts the movement
Message's a start
Moving
Start = true
Search the angle at the Table of Gaits. Apply the angle in the servo motor.
Loop
Start = false
Not moving
Still not moving

Figure 10. Receiver's algorithm at the Third Profile: unicast (blue), broadcast (red) and loop (yellow).


How many modules (1 to 10)? 5

What's the period size (1 to 32)? 10

Operating mode Custom

k= 1

alpha= 25

Initialize tables

modN1 modN2 modN3 modN4 modN5 modN6 modN7 m...

Address: 23

t	Ângulo	PWM
0	30	227
1	20	213
2	10	199
3	0	185
4	-10	171
5	-20	157
6	-30	143
7	-40	129
8	-50	115

Clean tables Convert Start!

Figure 11. Transmitter's program at the Third Profile.

2.3.2 Loss of Sync

The modules should, in theory, perform processing at a frequency of 6MHz, but the internal oscillators of each Atmel® ATmega 8® are different, then, with time, modules lose synchronization.

In modular robotics it's essential that there is timing in the movement, and, as the protocol control is open loop, the best solution found was to "restart" the system constantly, using the Christian's algorithm (Cristian (1989)). In it, one of the modules works as a "time server" and provides the time to reset the modules.

2.3.3 Reliability Packet Delivery Test

Figure 12 shows the execution of reliability package delivery test. Every interval Δt , the module 1 sends, a Standard Package with an angle message to module 2, which runs the First Profile. The packet address is different from module 2, then, the module receives it and transmits to the computer (via the RS-232), which counts the number of received packets. With different values of Δt we plotted graphs of transmission reliability percentage. The host sent 1000 packages in the test.

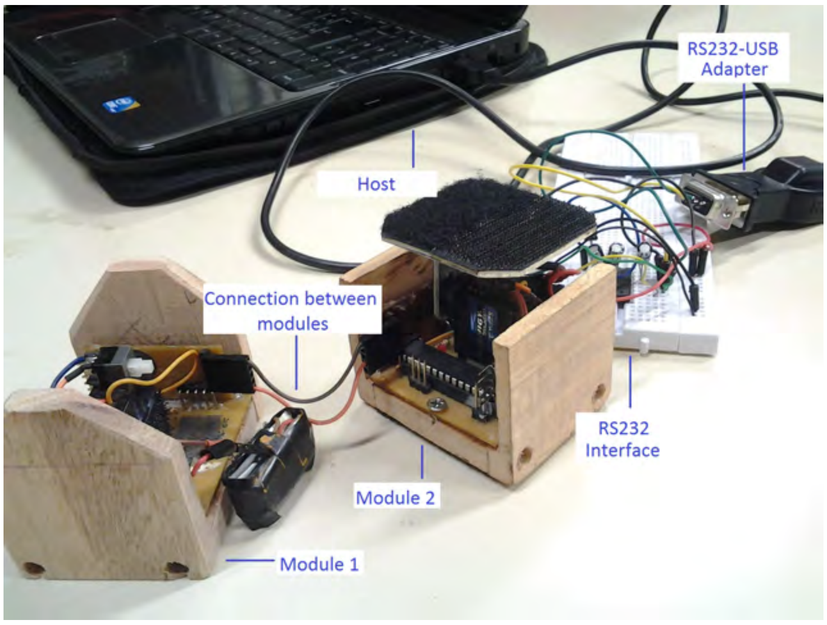


Figure 12. Preparation of Reliability Packet Delivery Test.

2.3.4 Estimation of Transmission Delay Test

As the topology of the protocol is a Line, there is a transmission delay proportional to the number of modules. In the test for estimating the transmission delay the host sends a packet to the modules, and they transmit back to the computer which reads the interval of time between sending and receiving a packet. Based on this time difference, we plotted graphs of the time delay by the number of modules in the system. The host sent 150 packages in the test.

3. RESULTS

3.1 Reliability Packet Delivery Test

The results of the reliability package delivery test (Fig. 13) shows that the protocol is effective for time intervals which are bigger than 200 ms.

3.2 Estimation of Transmission Delay Test

The results of the estimation of transmission delay test (Tab. 3) show that the delay time increases proportionally to the number of modules. With the use of c*ftool*, a tool from Matlab®'s tool , we performed a linear regression and found the linear polynomial function of Eq. (1), in which $delay_{total}$ is the total propagation delay and M is the number of modules in the system.

$$delay_{total} = 4.619M + 6.714 \tag{1}$$

In order to estimate the module delay, we divide Eq. (1) by the number of ErekoBot and calculate the limit when M tends to infinity, which results in a delay of $4.619\,ms$.

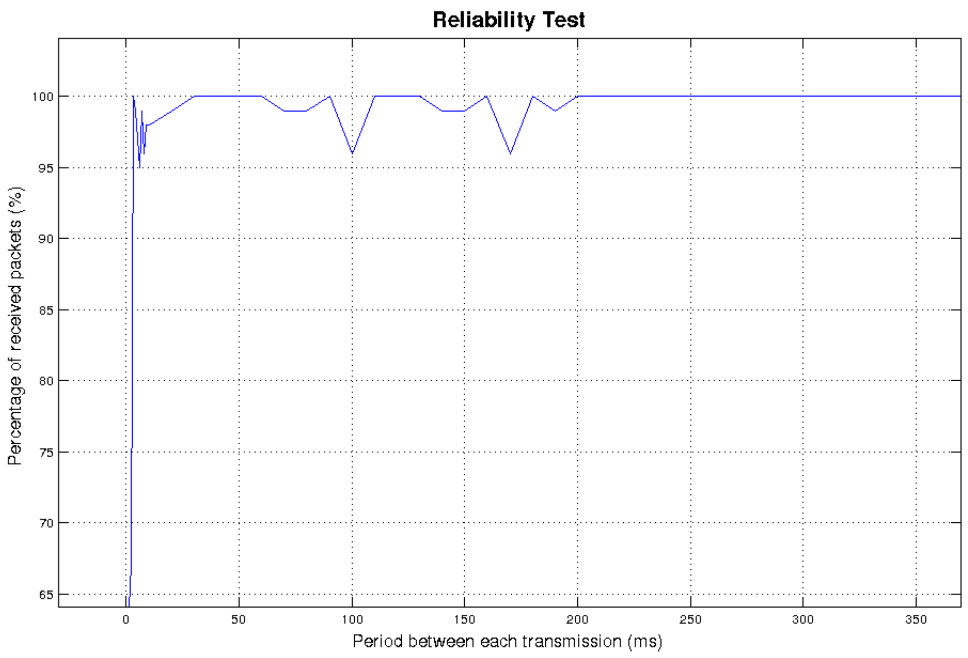


Figure 13. Tests results for packet delivery's reliability.

Table 3. Test results for estimating the transmission delay.

Number of Modules (M)	Delay (ms)
1	11
2	16
3	21
4	25
5	31
6	35
7	39
8	43

4. DISCUSSION

From the results of Fig. 13, we found that the packet delivery is efficient for time intervals greater than 200 ms. Considering that the ErekoBot movements already posses time limitation of the servo engines (they need at least 500 ms to stabilize between any positions [4]), Protocol data transmission realibility is enough for the system.

We also found that the delay per module is approximately 4.6 ms, which makes the number of modules become a limiting factor for moving synchrony.

With these two findings, we confirmed that the protocol can be used for ErekoBot, as long as movement specifications accept the lack of synchronization generated by the number of modules.

More recent work in Modular Robotics uses a centralized system (Arney *et al.* (2010)) (similar to the First and Second profiles), except in situations with sensors, which, in general, mix centralized with decentralized systems (Shen *et al.* (2002)) (similar to Third Profile). When these works do not deal with modular robots reconfiguration, they also use wiring communication (Hamann *et al.* (2010)).

The ErekoBot module is a simple robot and, therefore, the protocol meets its needs. For more robust modules, with automatic fittings and sensors, the protocol must be adapted.

A desirable update is the change of a topology in series for a bus one, which not only reduces the delay time, but also simplifies the system diagrams. Another possible direction is the implementation of wireless communication, so it does not hinder the movement of the robot.

However, for modular robotics, reconfiguration is the most important challenge, once from it, modules would be able to change their moving ways with user commands. The implementation of the Fourth Profile allows this reconfiguration

[4]Specification: http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=9392

in real time.

Obviously, being able to send and receive data from robots efficiently is important for the development of projects, but in the case of modular robotics, this importance is even greater because the host must deal with not only one, but several robots, which must react in a synchronized way at all times.

This work obtained a verified and validated function protocol, which may be useful to future projects in the area of modular robotics. This area promises a versatility that can radically change the automation, especially in places with limited reach.

5. ACKNOWLEDGEMENTS

The authors would like to acknowledge the Brazilian institutions: ANP, FINEP, MCT, and Petrobras for supporting the present study and the program PRH-PB 23.

6. REFERENCES

Arney, D., Fischmeister, S., Lee, I., Takashima, Y. and Yim, M., 2010. "Model-based programming of modular robots". In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*. pp. 66–74. ISSN 1555-0885. doi:10.1109/ISORC.2010.16.

Cristian, F., 1989. "Probabilistic clock synchronization". *Distributed Computing (Springer)*, Vol. 3, pp. 146–158.

Dept., E.I.A.E., 1969. *Interface between data terminal equipment and data communication equipment employing serial binary data interchange*. Electronic Industries Association. Engineering Dept.

Hamann, H., Stradner, J., Schmickl, T. and Crailsheim, K., 2010. "A hormone-based controller for evolutionary multi-modular robotics: From single modules to gait learning". In *Evolutionary Computation (CEC), 2010 IEEE Congress on*. pp. 1–8. doi:10.1109/CEC.2010.5585994.

Shen, W.M., Salemi, B. and Will, P., 2002. "Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots". *Robotics and Automation, IEEE Transactions on*, Vol. 18, No. 5, pp. 700–712. ISSN 1042-296X. doi:10.1109/TRA.2002.804502.

Souza, N.C.A., 2011. "Erekobot alfa project: Design and construction of a modular robot prototype". *21st Brazilian Congress of Mechanical Engineering*, Vol. 1.

Tanenbaum, A., 2002. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition. ISBN 0130661023.

Yim, M., 1994. "Locomotion with a unit-modular reconfigurable robot". Technical report.

Yim, M., Shen, W.M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E. and Chirikjian, G.S., 2007. "Modular self-reconfigurable robot systems – challenges and opportunities for the future". *IEEE Robotics and Autonomation Magazine*, Vol. March, pp. 43–53.

Yim, M., White, P.J., Park, M. and Sastra, J., 2009. "Modular self-reconfigurable robots". In *Encyclopedia of Complexity and Systems Science*, pp. 5618–5631.

Zhang, Y., Yim, M., Ackerson, L., Duff, D. and Eldershaw, C., 2004. "Stam: A system of tracking and mapping in real encironments". *IEEE J. on Wireless Communications*, Vol. 11, No. 6, pp. 87–96. URL `http://dx.doi.org/10.1109/MWC.2004.1368901`.

7. RESPONSIBILITY NOTICE