

Studienarbeit

Lucas Carvalho Cordeiro 1928 SA

**Development real time
applications with LEGO Mindstorms**

Stuttgart, 30th October 2003

**Tutors: Prof. Dr.-Ing. Dr. h. c. Peter Göhner
 M.Sc. Paulo Urbano
 Dipl.-Ing. Thomas Wagner**

Acknowledgements

The material in this these has been developed over the last three months and its results were driven in a valuable way.

I would like to thank my family who supported me over the last years to the completion of my studies.

I would also like to thank Prof. Göhner for giving me an opportunity to complete my undergraduate project at IAS. I am very thankful to my advisor Paulo Urbano for sharing his ideas and helping to complete my project.

Finally, I thank Ursula Habel for essential support throughout the UNIBRAL project and by the grammatical correction of my abstract in German. Especially thanks to CAPES for the scholarship offered over the UNIBRAL project.

Abstract

As the computers became smaller, faster, cheaper and more reliable their range of application is expanding. Computers are used to control a wide range of systems ranging from simple home appliances to entire manufacturing. These computers interact directly with hardware device controlling a physical, dynamic process. The software in these systems is an embedded real-time system that must react to events occurring in the control process and issue control signals in response to these events. It is embedded in some larger system and must respond, in real time, to changes in the system's environment.

The LEGO[®] MINDSTORMS[™] kit allows a wide variety of physical models to be built, which may be programmed via the RCX[™] processor integrated into them. Using its standard firmware, the RCX device may be programmed through several different specialist languages. However, the additional availability of bytecode-compatible replacement firmware for the RCX makes the use of languages such as C++ and Java[™] possible as the programming languages for this particularly attractive environment. The C++ and Java programming languages provide features to develop real-time applications as parallelism, synchronization, input and output of process values and bit operations. Within the scope of this research project, some of the issues associated with choosing an Integrated Development Environment to program MINDSTORMS models are investigated using C++ and Java within the context of real-time systems.

Zusammenfassung

Sowie Computer kleiner, schneller, billiger und zuverlässiger wurden, verbreiterte sich auch das Spektrum ihrer Anwendungsmöglichkeiten. Computer werden benutzt, um eine breite Palette von Systemen zu kontrollieren, die von einfachen Haushaltsgeräten bis hin zu ganzen Herstellungsabläufen reicht. Diese Computer interagieren direkt mit Hardware-Gerät, das einen dynamischen und physischen Prozeß kontrolliert. Die Software in diesen Systemen ist ein eingebettetes Echtzeitsystem, das auf Ereignisse reagieren muß, die im Kontroll-Prozeß auftreten und daraufhin Kontroll-Signale aussendet. Es ist eingebettet in irgendein größeres System und muß – in Echtzeit – auf Ereignisse aus der System-Umgebung reagieren.

Das LEGO[®] MINDSTORMS[™] Werkzeug erlaubt den Bau einer breiten Vielfalt physischer Modelle, die über den RCX[™] Prozessor, der in sie integriert wird, programmiert werden können. Das RCX Gerät kann durch mehrere verschiedene Spezialsprachen programmiert werden, indem man seine Standardfirmware benutzt. Jedoch macht die zusätzliche Verfügbarkeit von bytecode-kompatibler Ersatz-Firmware für den RCX die Verwendung von Sprachen wie C++ und Java[™] möglich, die als Programmiersprachen dafür besonders attraktiv sind. Die C++ und die Java Programmiersprachen stellen Merkmale bereit, um Echtzeit-Anwendungen wie Parallelismus, Synchronisierung, Input und Output von Prozeß-Werten und Bit-Operation zu entwickeln. Innerhalb des Bereichs dieses Forschungs-Projektes werden einige der Themen, die mit der Wahl, MINDSTORMS Modelle zu programmieren in Verbindung gebracht werden, erforscht. Dazu werden C++ und Java innerhalb des Kontextes von Echtzeitsystemen benutzt.

Table of Contents

Acknowledgements	2
Abstract	3
Zusammenfassung	4
1 Introduction	7
1.1 Motivation.....	7
1.2 General Purpose.....	7
1.3 Requirements to the software.....	7
2 Basics	9
2.1 Hardware and Firmware.....	9
2.1.1 The Robotics Invention System 2.0.....	9
2.2 Software.....	14
2.2.1 The RIS programming environment.....	14
2.2.2 LEGO Programming Languages.....	17
2.2.3 Introduction to real-time systems.....	19
3 Development of the Spring Mass System	26
3.1 Start Situation.....	26
3.2 Fundamental Design Decisions.....	29
3.3 System Architecture.....	30
3.3.1 System Overview.....	30
3.3.2 State Diagram.....	31
3.4 Software Components.....	32
3.4.1 Send Signal.....	32
3.4.2 Check Signal.....	37
3.4.3 Oscillate.....	39
3.4.4 Detect Obstacle.....	40
3.4.5 Play Sound.....	40
3.5 Test Specification.....	41
3.5.1 Introduction.....	41
3.5.2 Test Requirements.....	41
3.5.3 Test Methods.....	42
3.5.4 Test Criteria.....	42
3.5.5 Test Cases.....	43
3.5.6 Test results.....	43
4 Using the Software	44
4.1 Installation.....	44
4.1.1 Prerequisites.....	44
4.1.2 Accessing the embedded microcontroller of Robot.....	44
4.1.3 NQC Installation.....	45
4.1.4 BrickOS Installation.....	46
4.1.5 LeJOS Installation.....	47
4.2 Operation.....	49
4.2.1 Preparation of the robot.....	49
4.2.2 Running NQC programs.....	50
4.2.3 Running BrickOS programs.....	51
4.2.4 Running LeJOS programs.....	51
4.3 Control Elements.....	53
4.3.1 Batteries.....	53
4.4 Functions.....	54
5 Conclusion and Outlook	54

5.1	Summary	54
5.1.1	Introduction	54
5.1.2	Programming languages comparison to the LEGO Mindstorms	55
5.1.3	An application scenario with real-time characteristics	55
5.2	Experiences	56
5.3	Problems.....	56
Appendix A Index of Figures		58
Appendix B Index of Tables		59
Appendix C Abbreviations		60
Appendix D Terminology		61
Appendix E Literature.....		63
Appendix F Troubleshooting.....		65
F.1	I do not have access the embedded microcontroller over the IR Tower.....	65
F.2	The IR Tower still does not work	65
F.3	I cannot store other firmware	66
F.4	My compiler does not work	67
F.5	My robot does not work as desired	67

1 Introduction

1.1 Motivation

Real-time systems are becoming increasingly important in our society. There are many applications whose correctness is time dependent. They range from safety-critical systems such as nuclear reactors and automotive controller, to entertainment software such as games and graphics animation. Nowadays real-time computer systems replace many mechanical and hydraulic control systems with high requirements and dependability applications.

The microelectronic evolution and the CPUs becoming cheaper, smaller and more reliable can be used as computer systems in several appliances. These computers are different from our computers that we have at home, they are designed to perform a main task and it is not necessary to have a hard and floppy disk or keyboard. For instance, household, communication and maintenance devices only become functional using integrated microprocessor-based controls. A microprocessor-controlled wash machine is a good example, which prime function is to wash clothes, but it depends which clothes will be washed, then there are different “wash programs” that must be executed. These types of computer application are examples of **real-time** and **embedded**.

1.2 General Purpose

Apart from the standard software delivered with the Mindstorms kit, several independent groups have released development environments and operating system to the RCX microcomputer. The purpose of this undergraduate project was the investigation of the available methodologies, techniques and tools to develop real time applications based on the LEGO MINDSTORMS kit. In order to illustrate their characteristics, an application scenario with real time characteristics was defined and implemented with the different tools.

1.3 Requirements to the software

The LEGO MINDSTORMS kit comes with software that provides a graphical programming environment. Unfortunately, the software that comes with them is, although visually attractive, rather limited in functionality. Hence, it can only be used for simple tasks. To unleash the full power of the kit, we need a different programming environment. Therefore, the investigation of the available tools to develop real time applications based on the LEGO Mindstorms was extremely important.

The robot developed during this work has to perform several processes (tasks) in parallel and respond to stimuli that occur at different times. We have to consider carefully the synchronization of all this process, so that there are no interferences between them. The architecture must therefore be organized so that control is transferred to the appropriate handler for that stimulus as soon as it is received. In this sense, our software has a set of concurrent and cooperating process.

2 Basics

2.1 Hardware and Firmware

2.1.1 The Robotics Invention System 2.0

The core product of the LEGO Mindstorms series is the *Robotics Invention System (RIS)*. It intended to teach children and adults the basics of robotics using familiar LEGO Bricks. With the RIS we can build robots that move and react to inputs from their environment, e.g. touch and light. The robot's programs are written on a host computer, downloaded to the robot via infrared connection as depicted in figure 1 and then executed autonomously. The latter is probably the most fascinating regarding the Mindstorms – no cables or any other connection to a stationary computer is required for the robots to move around.

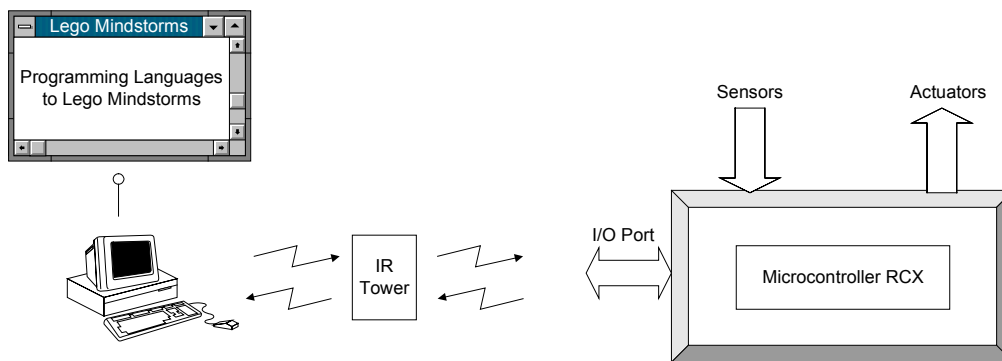


Figure 1 - System Interface

2.1.1.1 The RCX and the Lego firmware

The most important part of the Robotics Invention System is the Robotic Command Explorer (RCX). This is a special Lego brick with an integrated Hitachi H8/3292 microcontroller (running at 16 MHZ), three sensor inputs, three motors outputs, an IR transceiver, an LCD, and four control buttons as depicted in figure 2. A simple speaker for sound output is also integrated. The RCX requires six batteries of type AA to run.

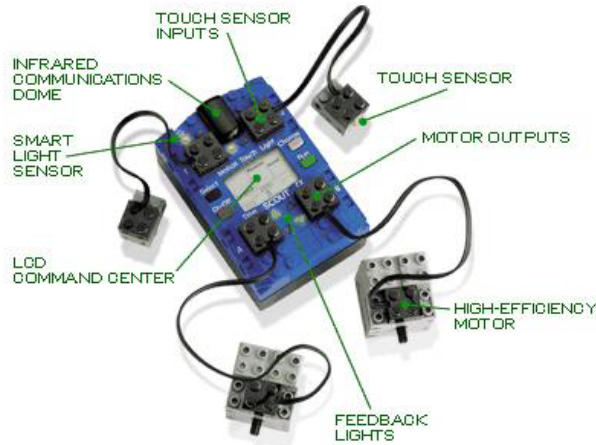


Figure 2 - RCX™ Microcomputer

Besides low-level hardware access routines, the 16 KB on-chip ROM of the RCX only provides basic communication logic to download a *firmware* that serves as the runtime environment for any user-written programs. Additionally, the RCX contains 32KB of RAM that is shared by the firmware and the actual programs.

The following pages describe the features of the RCX in the context of the current LEGO firmware (version 2.0). Most of the features are available in other execution environments as well, but organized differently.

Besides the RCX, LEGO offers two other programmable bricks (short P-bricks) called *Cyber-Master* and *Scout*. The latter is included in the *Robotic Discovery Set*. Of all P-bricks, the RCX is the most advanced.

Commands and programs The LEGO firmware is basically a byte code interpreter. A user program must be downloaded into one of five *slots* (using the IR interface) before it can be executed by the firmware. All byte codes are documented in the RCX 2.0 Beta SDK[1]. Besides downloading a complete program, individual commands can also be executed directly via the IR link.

All commands consist of an opcode (1 byte) and, whether required, parameters (up to 5 bytes). The RCX secures every command transmission by performing a logical not operation on the command byte and sending back the result (except for bit 4 which has a special meaning). Downloads of firmware and programs are secured by simply adding up the bytes and performing a module operation on the result (mod 256 for programs, mod 65536 for firmware) [26].

Task, resources and events One of the outstanding features of the RCX (or rather the LEGO firmware) is built-in multitasking. Up to 10 concurrent tasks per slot are possible. The priority of each task can be adjusted between 0 (highest) and 255 (lowest).

Since multitasking can lead to conflicts – e.g. when two tasks want to play sound at the same time, the LEGO firmware manages four resources: the three motors outputs and sound output. Each task can request access to these resources. Access is granted if no task of higher priority owns the resources in question. Access is denied (or taken away if granted before) as soon as a task with equal or higher priority requests the same resources. A task can specify a handler routine that is executed when it loses any of its acquired resources, so it can act appropriately. It should be noted that access control is not mandatory. Whether a low priority task uses motor commands without applying for access first, these commands are executed even if another task currently “owns” the motors.

A task can also monitor up to 16 different events. These events are freely configurable, i.e. we can specify which event source to monitor and what type of event we want to receive [26]. The possible event sources are:

- A physical sensor (touch, light, rotation, temperature or self-made).
- A timer.
- A counter.
- An incoming IR message.

In addition to the event source, we must specify what of change should trigger the event. For example, whether we specify `EVENT_TYPE_LOW`, the event is triggered when a (configurable) lower limit is reached. The firmware that comes with the LEGO Mindstorms kit supports 11 different event types, including clicks (a value goes from low to high and back to low), rapid changes, and incoming IR messages. To receive events, a task specifies an event handler that interrupts normal execution as soon as an event is triggered.

Sensor inputs As already mentioned, the RCX has three sensor inputs. We can connect one or more sensors to each input. Whether more than one is connected, we may or may not be able to differentiate between them [2], so the usual case is having one sensor on one input. Sensors are connected to the RCX with special cables that have small Lego bricks with metal contacts at both ends. Some sensors (e.g. the light sensor) have cables where one end is hardwired to the sensor.

There are five types of sensors that can be used with the RCX (this number may grow with future firmware versions): touch, light, temperature, rotation and generic. Generic means

that, unlike the other types, the sensor signal is not pre-processed in any way. This is useful for self-made sensors. Sensor values can be used as event resources or queried directly.

Timers and counters The RCX has four timers with 10 ms resolution. Timers can be cleared (i.e. set to 0), set to a specific value, queried directly, and used as event resources. Counters are similar to variables. However, counters can only be incremented or decremented by one. What makes them interesting is that they can be used as event sources.

Motor outputs The RCX can drive motors through its three 9V outputs. Only one motor can be connected per output. The behaviour of the motors is determined by the mode, direction, and power attributes. Mode can be one of *on*, *off* or *float*. “On” means the motor is turning with the current power setting, “off” means the motor actually breaks, and float (switches the motors smoothly) lets it more freely, but without turning on its own. The motor direction can be *forward* and *reverse*. The power of a motor is a constant between 0 (low) and 7 (high).

The RCX’s three outputs are independent, so the attributes just described can be specified per motor. For example, a robot with one motor on each side can turn around by setting one to *forward* and one to *reverse* direction.

Motors can also be globally disabling, set to a specific direction, and limited regarding their maximum power. Whether the motors is disabling, none of the regular motor commands has an effect. Whether the motors are globally set to reverse direction, all direction commands are interpreted inversely, i.e. forward as reverse and reverse as forward.

Besides motors, other actuators can be connected to the outputs as well. For instance, Lego offers a small “lamp” the brightness of which can be regulated through the output power. Naturally, directional settings do not make sense in this case.

The IR transceiver Communication from the RCX to a host computer or another RCX is realized through the integrated IR transceiver. The supported baud rates are 2400 and 4800. Other settings that can be specified programmatically include sending power (long vs. short range), carrier frequency, and packet format. Details concerning the IR protocol can be found in [3].

Besides uploads and downloads from and to the host, the firmware supports the exchange of simple messages between RCX bricks where each message consists of a single byte. Received messages are stored in a 1-entry buffer from where the currently running program can read them. Another message can only be received when the program has cleared the buffer.

In [2], Mark Overmars also shows a way to combine the IR transceiver and a light sensor to get a rough proximity sensor. Unfortunately, there is no other way to measuring distances, except with self-made sensors.

Miscellaneous In addition to the features already mentioned, the RCX supports sound generation. First, there are built-in system sounds (like frequency sweeps up and down). The second way of producing sound is by specifying a frequency/duration pair.

What is really annoying concerning the current LEGO firmware version is its enormous size. If you subtract the 24 KB of the firmware from the available 32 KB, only 8 KB are left for own programs. Also, performance can be an issue at times since all commands are interpreted [26].

Finally, there is an integrated watch in 24-hour format. Probably its only useful purpose is being displayed on the LCD.

2.1.1.2 LEGO Technique parts and sensors

Most of the RIS consists of standard Lego Technique parts. Only the two included motors seem slightly different, and there are a few special parts some of which are shown in figure 3. Additionally, the RIS contains two touch sensors and one light sensor that can be connected to the RCX.



Figure 3 - Special Mindstorms parts

With more than 700 pieces, one RIS set already allows for rather complex robot constructions. Whether that is not enough, one can always purchase additional Lego Technique sets or one of the Mindstorms Expansion sets (see the LEGO Mindstorms Homepage [4]). More and different sensors can be purchased individually, most notably a rotation and a temperature sensor.

2.1.1.3 The IR Tower

An important part of the RIS is the so-called *IR Tower*, which serves as a communication facility between a PC and the RCX. Like the RCX transceiver, the tower has two ranges setting (near and far). With both the tower and the RCX set to far, we have been able to cover a distance of 1.6 meters.

In the 2.0 version of the RIS, the IR Tower is also available in Universal Serial Bus (USB). The great advantage is that the USB provides enough power to run the tower without a battery.

2.1.1.4 LEGO Mindstorms Robotic Invention System Costs

The table 1 depicts the Robotic Invention system prices in the time I was writing my undergraduate project.

Table 1 - LEGO Mindstorms Robotic Invention System (RIS) 2.0 Costs

Company	Price	Country
HobbyTron.com	\$ 189.00	USA
Amazon.com	\$ 199.99	USA
KBToys	\$ 199.99	USA
DesignTyme Toy Store	\$ 199.99	USA
MyToys.de	€ 249.99	GER
SpielShop.de	€249,99	GER

2.2 Software

2.2.1 The RIS programming environment

The following sections are based on version 2.0 of the RIS [1]. The Software Development Kit (SDK) contains:

- ✓ Protocol information for the new USB IR communication tower (select Custom installation);
- ✓ API headers and documentation for the new Windows PC based communication stack, Ghost;
- ✓ Complete documentation for all LEGO Assembly (LASM) commands;
- ✓ Interactive help for LASM & MindScript programming;
- ✓ A dual-interface COM server component that wraps Ghost, LASM & MindScript;

- ✓ A number of example programs in Microsoft Visual Basic and Visual C++;
- ✓ An example IDE for writing LASM & MindScript programs.

2.2.1.1 - User Interface

The software that comes with RIS provides a complete environment for programming Mindstorms robots. The core of the RIS software is an interactive visual flow charter shown in figure 4. LEGO bricks represent actions (e.g. turning on a motor), sensors, and flow control (like conditional processing or loops). The whole user interface is designed for ease of use, including step-by-step tutorials with videos accompanying each step.



Figure 4 - The Robotics Invention System programming environment

Sensor inputs can be processed with so called *sensor watchers*. The light sensor watcher, for example, has two connection points – one for dark and one for bright values. The commands under the dark connection point are executed whenever the sensor value enters a configurable “dark” range. The commands under the light connection point are executed whenever the sensor value enters a configurable “light” range. There is a test panel will let you take readings from your sensors and even test motors attached to the RCX unit. Using the test panel you can calibrate specific values for your robot to respond to using up to 3 sensors. The figure 5 shows a measured that represents the value of light falling one the light sensor attached to the RCX unit.

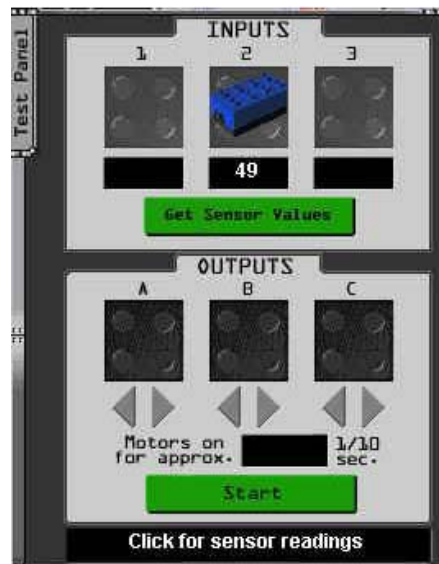


Figure 5 – Test Panel shows the value measured of light sensor

The RIS software only provides a single counter that can just be incremented and reset to 0 again. The only way to react to the counter’s value is a sensor watcher with a configurable range of values and a single connection point. Whenever the counter enters the specified range, the commands under the connection point are executed. The RIS software has no support for variables.

Only one of the RCX’s timers is available in the RIS environment, and only with 100 ms resolution. All we can do with the timer is reset it and use it as an event source (represented by the timer watcher). Commands that should be executed directly after pressing the Run button on the RCX – rather than waiting for an event - must be placed under the “program” brick.

2.2.1.2 Generated programs

The programs generated from the flow chart consist of three kinds of tasks:

- ✓ The main task initializes timer and counter, the sensor inputs, and the motor outputs. Afterwards, it continuously monitors the events;
- ✓ The program task executes the commands placed under the “program” brick. It is started by the main task after initialisation is complete;
- ✓ One task per sensor watcher connection point. As soon as the corresponding event is triggered, the task is started. If it is still running when the next event arrives, it is restarted abruptly.

2.2.1.3 Additional features regarding the Firmware RCX 2.0 BETA SDK

There are several features included in RCX 2.0 firmware (version 03.28) and the most noticeable are indicated below:

- ✓ **Event monitoring** on physical and a number of virtual sensors (timers, IR message and counter variables);
- ✓ **Priority-based access control** to shared resources;
- ✓ **Local variable tasking**, for example, safe parameter passing to subroutines that execute within the environment of the calling task;
- ✓ **Global motor and sound control**;
- ✓ **Play tones** with the frequency taken from a variable;
- ✓ **Display and track the value of internal RCX data** with or without the program running. (Great for demonstrating or debugging.);
- ✓ The ability, under program control, to **switch to another program slot and start it running** - just as you can with the LEGO MINDSTORMS Remote Control Unit;
- ✓ **Simple support** for data arrays/variable pointers.

2.2.2 LEGO Programming Languages

Apart from the standard software delivered with the Mindstorms kit, several independent groups have released development environments and operating system to the RCX microcontroller. The next three subsections present a short description concerning the programming languages.

2.2.2.1 Not Quite C

NQC created by Dave Baum [5] stands for Not Quite C, and is a simple language for programming several LEGO Mindstorms products. Some of the NQC features depend on which Mindstorms product you are using. This product is referred to as target for NQC. Nowadays, NQC supports four different targets: RCX, CyberMaster, Scout, and RCX2 (an RCX running 2.0 firmware).

The pre-processor and control structures of NQC are very similar to C. NQC is defined as two separate pieces. The NQC language describes the syntax to be used in writing programs. The NQC API describes the system functions, constants, and macros that can be used by programs. This API is defined in a special file built in to the compiler. By default, this file is always processed before compiling a program.

The RCX has a Hitachi H8 microcontroller with 32 KB of external memory. An on-chip, 16 KB ROM contains a driver that is run when the RCX is first powered up. The on-chip driver is extended by downloading 16 KB of firmware to the RCX. Thus, NQC allows user programs are downloaded to the RCX as byte-code and stored in a 6 KB region of memory.

2.2.2.2 Brick Operating System

BrickOS created by Markus [6] is an open source embedded operating system and provide a C and C++ programming environment to the LEGO Mindstorms Robotics kit, allowing the use of the languages such as C and C++ instead of the standard LEGO programming Language. BrickOS has a multi-tasking system running on the RCX hardware and providing an API to all the RCX sensors and actuators.

BrickOS consists of an alternative operating system for the Mindstorms RCX and demonstration programs written in C and C++. The operating system (OS) and the compiled programs are downloaded to the RCX and executed autonomously. The brickOS distribution provides the sources for the operating system, demo programs and utilities.

BrickOS allows running the RCX in native mode¹ (the data-structure is represented by a series of logical records and do not need to save data in files); this means that you can use full 32 KB memory.

2.2.2.3 LeJOS Virtual Machine

LeJOS² created by Solorzano [7] is an implementation part of the Java Virtual Machine (JVM) running on the RCX hardware. LeJOS comes with firmware that replaces the original RIS firmware; the new firmware implements many core Java classes. The JVM is analogous to the standard LEGO firmware. It is the component that must first be uploaded to the RCX to accommodate Java programs.

Like any programming language, it allows you to control the program flow. The LEJOS API mirrors the basic functionality of LEGO code, controlling motors, sensors, and other elements of the RCX brick. LeJOS includes a trimmed-down version of the standard Java API with general classes to help out with programming as the vector class.

As mentioned before, the RCX has a total of 32 KB of RAM (figure 6) and about 4 KB is off limit because the ROM routines use it. That leaves about 28 KB free to be

¹ The native mode is complemented in BrickOS by one "emulator mode".

exploited. LeJOS has a footprint of about 16 KB, which means there is a total of 12 KB of free memory for user code. That number is usually more than enough for most robotics programs, unless the RCX is heavily collecting and storing data, such as navigation.

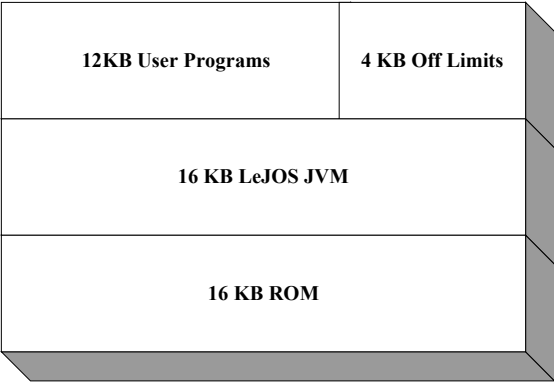


Figure 6 - The LeJOS JVM memory map

2.2.3 Introduction to real-time systems

This section is concerned with definitions, examples and characteristics of real-time systems. It studies the particular characteristics of these systems and remarks are made regarding the real-time applications.

2.2.3.1 Definition of real-time systems

There are many definitions of real-time systems, nevertheless they have in common the notion of response time. The *Oxford Dictionary of Computing* has the following definition of real time-system: “*Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to related to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness*”.

The Institute of Industrial Automation and Software Engineering (IAS) has defined ‘real-time’ programming as: *creation of programs in such a way that the **time requirements** on the **compilation of input data**, on the **processing** and on the **delivery** of output data are fulfilled* [17].

Experts in the field of real-time computer system classify two categories of real-time system: **hard** and **soft** real-time system. Hard real-time systems are those where it is

² Some authors consider the name LeJOS as a play of word on LEGOs, with “g” replaced by “j” for Java. Other authors consider as Java Operating System.

absolutely imperative that responses occur within the specified deadline³ [14]. A typical example is an autopilot on an aircraft, the failure of which to check the altitude at prescribed times may have catastrophic consequences. Soft real-time systems are those where response times are important but the system will still function correctly if deadlines are occasionally missed [14]. A typical example of soft is a data acquisition system with many sensors for a process control in industry, because sampling of input sensor in a single activity has regular intervals but tolerate delays.

2.2.3.2 Generalized embedded computer system

In general, real-time systems for management and controlling of processes need an interface between the real world and the computer. Thus, there is Analog to Digital Converter (ADC) to convert the analog signal from sensor to digital signal. A real-time clock is necessary because the input sensor signal must be sampled in a regular interval [14]. Algorithms for digital control will process the information (in the most of cases involving complex calculus) from sensors and will record in a database system. All information concerning engineering system is recorded in a database and the operator’s console can see this information and interrogate. Indeed, there is a support decision making in the day-to-day running systems. For instance, in process industries, plant monitoring is essential for maximizing economic advantages rather than simply maximizing production [14]. The figure 7 shows a typical embedded system. The software consists of many modules, which contains the algorithms for physically controlling the devices, a module responsible for recording the system’s state changes, a module to retrieve and display those changes and a module to interact with the operator.

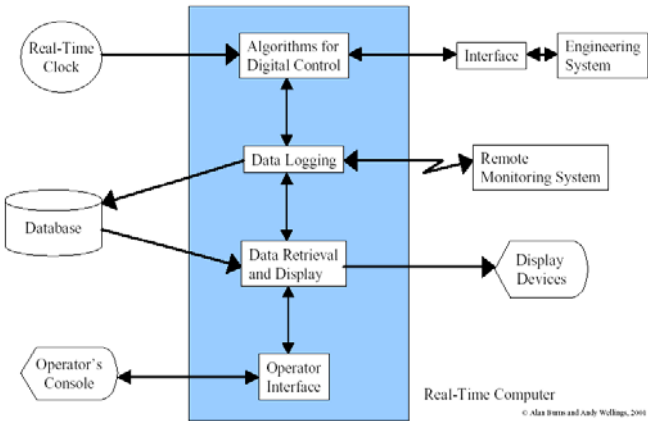


Figure 7 – A typical embedded system

(Source: <http://www.cs.york.ac.uk/rts/RTSBookThirdEdition.html>)

³ A deadline is either a point in time (time-driven) or a delta-time interval (event-drive) by which a system action must occur.

2.2.3.3 Characteristics of real-time systems

Most real-time computer systems are parts of other systems, including larger computer systems. The final effect of the computer system would be through mechanical or other systems interacting with the environment. A real-time system may therefore be viewed as consisting of the *controlling system*, usually a computer, and the *controlled system*, which in turn consists of *sensors* and *actuators* [18]. The system receives information about the environment from sensors and affects the environment by means of actuators. Some actuators may not necessarily act on the environment, but, instead, may simply adapt to changes in the environment in a passive manner, for example by controlling the system exposure to the environment or by activating additional sensors. Sensors are operated by the effect of light, heat, pressure, temperature, current, voltage, radar and others forms of energy in the outside environment, and take the form of pressure gauges, thermometers, microphones, hydrophones, receivers and video cameras. The sensors may produce analog or discrete data. Actuators may effect the environment through discharging similar forms of energy to the environment in a controlled manner and may take the form of mechanical devices, heaters, displays, switches, transmitter, etc.

The system interaction with the environment may be both periodic and sporadic. This results from the fact that every real-time system necessarily consists of a monitoring component and a reactive component. Real time systems not possessing both these components in some form are rare and are of limited purpose. An example of a real-time system that is dedicated primarily to monitoring is a seismograph, an instrument intended for detecting and recording ground motions due to earthquakes in real-time. In general, the monitoring subsystem is intended for sampling the environment and, therefore, may function basically in a highly regular predictable manner. The reactive subsystem, on the other hand, may be called into action sporadically upon detection of irregularities or excessive deviations in the controlled or observed environment by the monitoring subsystem [18]. In this sense even a seismograph has a reactive component.

A. Timeliness

The timeliness of an action has to do with the action meeting time constraints, such as deadline. Deadlines may be hard or soft. Missing a hard deadline constitutes a system failure of some kind, so great care must be taken to ensure that all such actions execute in a timely way [15].

The basic concepts of timeliness in real-time systems are straightforward. Actions must be begun in response to event arrival or due-time arrival, and they must complete within

a certain time after they begin. These actions may be simple digital actuations, such as turning on a light, or complex loops that control dozens of actuators simultaneously. Typically, many subroutines or tasks must execute between the causative event and the resulting system action. External requirements bound the overall performance of the control path. Each processing activity in the control path is assigned a portion of the overall time budget. The sum of the time budgets for any path must be less than or equal to the overall performance constraints⁴.

B. Responsiveness

Virtually all real-time systems connect to either monitoring or controlling hardware, or both. Sensors provide information to the system about the state of the external environment. Many real-time systems use actuators to control their external environment or to guide some external processes. Flight control computers command engine thrust and wing and tail flap orientation to meet flight parameters. Chemical-process-control systems control when, what kind, and the amounts of different reagents added to mixing vats [15].

Naturally, most systems containing actuators also contain sensors. Although there are many open-loop control systems, the majority of control systems use environmental feedback to ensure that the control loop is acting properly. Standard computing systems respond primarily to the user⁵. Real-time system, on the other hand, may interact with users, but they have more concern for interactions with their sensors and actuators. One problem that arises with environmental interaction is that the universe has an annoying habit of disregarding our opinions of how and when it ought to behave [15]. External events are often not predictable. Nevertheless, the system must react to events when they occur rather than when it might be convenient.

C. Correctness and Robustness

A system is *correct* when it does the right thing all the time. Such a system is *robust* when it does the right thing under novel (unplanned) circumstances, even in the presence of unplanned failures of portions of the system [15]. Naturally, correctness and robustness are considered good things, but achieving them in a complex design is anything but trivial.

⁴ This includes initiation, preemption (postponement of execution by a high-priority task, including interrupts) and blocking times (prevention of the execution of a higher-priority action by a lower-priority action), if applicable

⁵ It is true that behind the scenes, even desktop computers must interface with printers, mice, keyboards, and networks. The point is they do this only to facilitate the user's whim.

D. Concurrency

Concurrency is the simultaneous execution of multiple sequential chains of actions. These action chains may execute on one processor (pseudo-concurrency) or multiple processors (true concurrency). Issues surrounding the execution of concurrent systems have to do with the scheduling characteristics of the concurrent thread, the arrival patterns of incoming events, the rendezvous patterns used when threads must synchronize, and the methods of controlling access to shared resources [15]. These can be nontrivial issues to deal with, particularly when one must consider the performance, as well as the functional requirements, of the system.

D.1 The Rendezvous Concept

If concurrent threads were truly independent, life would be much less interesting. Fortunately, threads must communicate to synchronize control or share resources. Communication in object systems takes place via messages. Messaging is a logical abstraction that includes a variety of rendezvous patterns, such as synchronous function call, asynchronous, waiting, timed, and balking [15].

A telephone is a better analogy for synchronous communication. The sender now waits until contact is made and the identity of the receiver verified before the message is sent. If the receiver can reply immediately (that is, during the same call), the synchronization is remote invocation. Because the sender and receiver ‘come together’ for a synchronized communication it is often called a **rendezvous** [19].

D.2 Semaphores

Edsger Dijkstra [16] created the *semaphore* as one solution to the protection of critical sections⁶. A semaphore is essentially a “lock” that serializes access to the resource it protects. Dijkstra used the p() and v() operations to allow tasks to grab the resources if the semaphore is free and to relinquish ownership of it, respectively. The p() operation returns only when the resource is available; otherwise, it waits. Once the resources is available, the p() operation locks the resource and call returns. The v() operation releases the lock and relinquishes ownership. Semaphores are simple, lightweight abstractions, which may not provide the behaviour richness required for more-complex resource sharing.

⁶ The code that must execute without interruption to avoid deadlocks and race condition

E. Sharing Resources

Another common problem in concurrent systems is the robust sharing of resources. Most correct solutions involve the serialization of access through mutual exclusion semaphores, or queues [15]. In object systems, such access control may be done through semaphore pattern. This can be indicated using the pattern notation of the modeling language (if supported) or by adding {guarded} constraints on the relevant operation. Data corruption can occur whenever there is nonatomic access to a data structure within a concurrent architecture.

F. Predictability

A key aspect of many real-time systems is their predictability. This is a crucial for many safety-critical and high-reliability systems, such as nuclear power plants, avionics systems, and medical devices [15]. The predictability of a system is the extent to which its response characteristics can be known in advance. This can be determined, in some cases, by static mathematical analysis, such as rate monotonic scheduling (RMA). In other cases, it can be ensured by disallowing pre-emption and using simple control algorithms, such as cyclic executives. Using active objects to represent tasks and identifying the performance characteristics of the tasks allows a variety of scheduling algorithms to be used. Another aspect of predictability is with respect to memory. There are two orthogonal views of memory: *usage* and *persistence*. In terms of usage, memory is typically thought of as divided into several categories:

- Execution memory, where the executable code resides
- Data memory (Stack, Heap, Static)

In terms of persistence, most real-time developers categorize memory into:

- Non-writable persistent
- Writable persistent
- Volatile

In terms of predictability, the areas of concern are the stack and the heap. The stack is the simplest. The developer must ensure only that the amount of stack space is sufficient to hold the “automatic” variables and return address in all cases [15]. The stack is managed at run-time by (compiler-generated) code as part of the protocol of subprogram calls [22]. This

is relative simple to do. The heap is more challenging for a variety of reasons. Heaps usually work by requesting a block memory from the operating system (through the *malloc*, or new operators of the programming language) [15]. The heap is managed by the run-time system or else by explicit user-provided code, reacting to allocation and deallocation requests [22]. Most heapers managers do not require a constant (or even bounded) time to allocate a block of memory. This is because most heap managers must traverse the free store to find a piece of memory that can fulfill the pending request. This means that deadline can be missed.

A more insidious problem is *memory fragmentation*. Most heaps are nothing more than large blocks of memory out of which blocks of any size can be allocated on a first-come-first-served basis. Through the process of repeated allocation and deallocation, the free memory heap can become fragmented into many small-isolated block. In such case, should a request come in for a block that is bigger than any available free block, the request must be denied, which can lead to catastrophic consequences [15].

2.2.3.4 Comments concerning the real-time applications

A reactive or event-driven system is one whose behaviour is primarily caused by specific reactions to external events rather than being self-generated. A time-driven is one whose actions are driven primarily by either the passage of time or the arrival of time epochs. Time-driven systems are those primarily driven by periodic tasking rather than by the arrival of periodic events [15].

A hard real-time system is required to produce the intended result before a specified point of physical time, the deadline. This point of time is determined by the application the computer system is intended to service. The controlling real-time software must be designed to generate the correct behaviour of the computer both in the value domain and in the temporal domain to meet this application requirement [20]. A soft real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements.

A schedulable system is one that can be guaranteed to meet all its performance requirements [15]. A task⁷ is an encapsulated sequence of operations that executes independently of other tasks. In a multitasking system, tasks are scheduled to run via a scheduling policy. Most real-time systems use the priority of a task to control when it runs in relation to other tasks that are ready to run. The priority of a task depends on the urgency of the task completion and its importance. If multiple tasks are ready to run, then the task with

the highest priority⁸ will execute preferentially. The scheduling policy of tasks may be either event-driven (that is, the task depends on a awaited event) or time-driven (that is, the task waits for its next scheduled start time to arrive). As a practical matter, most event-driven systems respond to discrete events whose time is relatively unpredictable, while most time-driven system deal with continuous control systems.

3 Development of the Spring Mass System

3.1 Start Situation

The general purpose of this undergraduate work was the investigation of the available methodologies, techniques and tools to develop real time applications based on the LEGO Mindstorms kit. In order to illustrate their characteristics, an application scenario with real time characteristics was defined and implemented with the different tools.

Based on this purpose, this work can be divided into three parts, first we will investigate the available methodologies and techniques based on the LEGO Mindstorms, second we will investigate the available different tools to develop real time applications and finally we will illustrate the methodologies, techniques and tools investigated with an application scenario with real time characteristics.

The analysis method *Use Cases* has been used. In this context, one external actor is present and the figure 8 shows a simplified diagram.

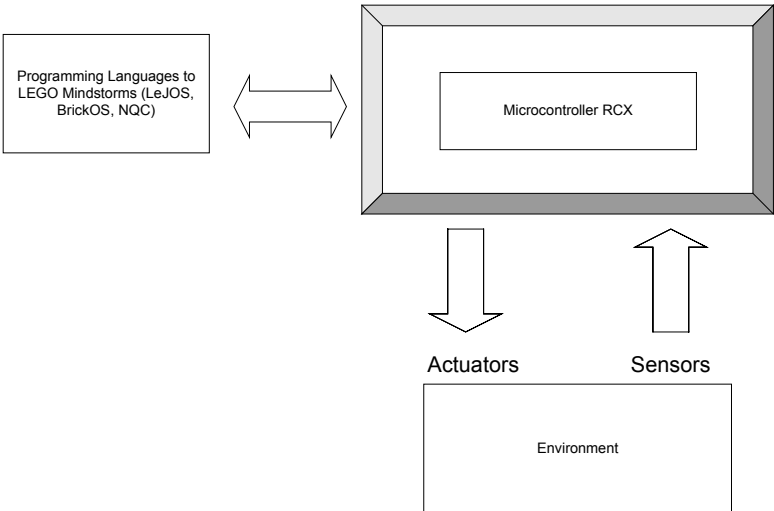


Figure 8 - A simplified diagram

⁷ Some authors differentiate between a task (very encapsulated) and a thread (less well encapsulated). These may be different in the coding level because they invoke different RTOS services.

⁸ Some RTOS treat the higher numerical value as the higher priority while others treat the lower numerical value as a higher priority.

The three programming languages to LEGO Mindstorms as described in section 2.2.2, will download the programs to the microcontroller RCX and will be stored in a 6K region of memory (depending of the sort of platform). The microcontroller will be used to control two motors, one light sensor, one touch sensor and an infrared port. The unique actor named “Environment” will interact with our application.

The application with real time characteristics will be develop similar the Hooke’s Law for a simple spring-mass system. The motion of a body that oscillates back and forth is defined as simple harmonic motion if there exists a restoring force F that is opposite and directly proportional to the distance x that the body is displaced from its equilibrium position [21]. This relationship between the restoring force F and the displacement x may be written as

$$F = -kx \quad (1)$$

where k is a constant of proportionality. The minus sign indicates the force is oppositely directed to the displacement and is always towards the equilibrium position. Whenever the body is subject to such a restoring force and is caused to oscillate back and forth, the time for one complete oscillation is defined as the period T and will be given by

$$T = 2\pi\sqrt{\frac{m}{k}} \quad (2)$$

the frequency f of oscillations is the number of oscillations per unit time and is the reciprocal of the period, $f = 1/T$, and is given by

$$f = \frac{1}{2\pi}\sqrt{\frac{k}{m}} \quad (3)$$

The maximum distance that the mass is displaced from its equilibrium position is called the amplitude of the oscillation. In simple harmonic motion the period and frequency are independent of the amplitudes of the oscillations.

There are a number of examples of bodies that undergo simple harmonic motion, and one of the simplest is a mass suspended and set to vibrating on a spring [21]. When the mass is hung on the spring and then displaced from its equilibrium position, it will oscillate up and down. The motion of the mass will be simple harmonic motion because the spring supplies a

force that is directly proportional to the displacement. The restoring force supplied by the spring is always directed back towards the equilibrium position regardless of whether the mass is above or below the equilibrium position. The constant of proportionality k is called spring constant and can be found by subjecting the spring to an applied force and measuring the amount that the spring stretches. The general property of matter, and springs in particular, whereby an applied force F_a , causes a displacement x from the equilibrium position that is directly proportional to the force, is known as Hooke's Law [21]. The figure 9 exemplifies the Hooke's Law. Hooke's Law may be stated as

$$F_a = kx \quad (4)$$

and may be used to calculate the spring constant k . For equal displacements, the applied force and the restoring force are equal and opposite. Therefore, Hooke's Law provides the conditions for simple harmonic motion and the spring constant can be used with Equation (2) to measure the period of a mass vibrating on a spring.

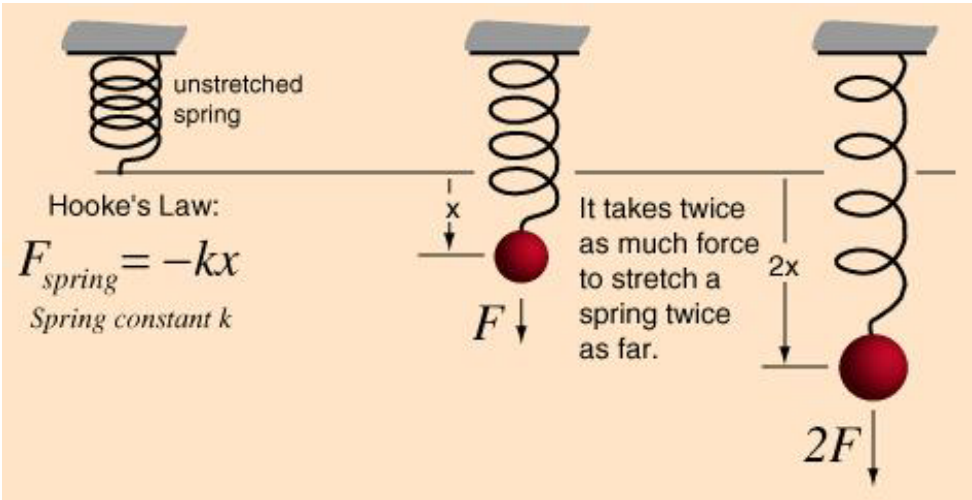


Figure 9 - Hooke's Law

The robot will be the object connected the spring and the spring base will be a unit with reflection properties. The spring will be imaginary and the spring constant (k) will undergo a value defined by us.

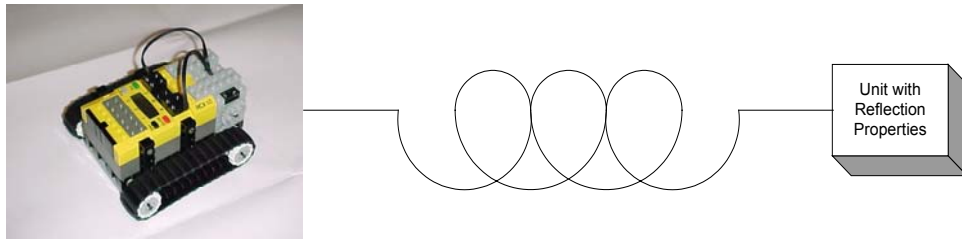


Figure 10 - Spring-Mass System with LEGO Mindstorms

The infrared port will send signals periodically and the light sensor connected in front of the robot will detect the fluctuations as well as possible. A unit with reflection properties will cause the fluctuations detected by light sensor. When an object with reflection properties is closer than the base distance the robot will drive backwards with the use of two motors coupled and after will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation. Whether an object with reflection properties is more distant than the base distance, the robot will drive forwards and after will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation. Whether there is an obstacle behind the robot the touch sensor connected behind them will stop the execution of the process and will generate sounds until the obstacle is removed.

The spring-mass system with Lego Mindstorms will perform a soft real time system in which there are no explicit deadlines. The data acquisition system will measure the signals from the sensors at regular intervals but to tolerate intermittent delays.

3.2 Fundamental Design Decisions

As we already know, an application scenario with real time characteristics must be defined and implemented with the different tools. This application implies in activities (tasks) that will be executed in parallel. The infrared port will send signals periodically and the light sensor connected in front of the robot will detect the fluctuations as well as possible. When an object with reflection properties is closer than the base distance the robot will drive to backward until no fluctuations have been detected and then will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation. Whether an object with reflection properties is more distant than the base distance the robot will drive to forward until no fluctuations have been detected and then will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation. Whether there is an obstacle behind the robot the touch sensor connected behind them will stop the execution of the process and will generate sounds until the obstacle be removed. All these described functions will correspond to software components.

The software components that realize all functionalities are described below:

- **Send signal** component;
- **Check signal** component;
- **Oscillate** component;
- **Detect obstacles** component;
- **Play sound** component.

3.3 System Architecture

3.3.1 System Overview

In the following diagram, an overview of the system architecture is presented.

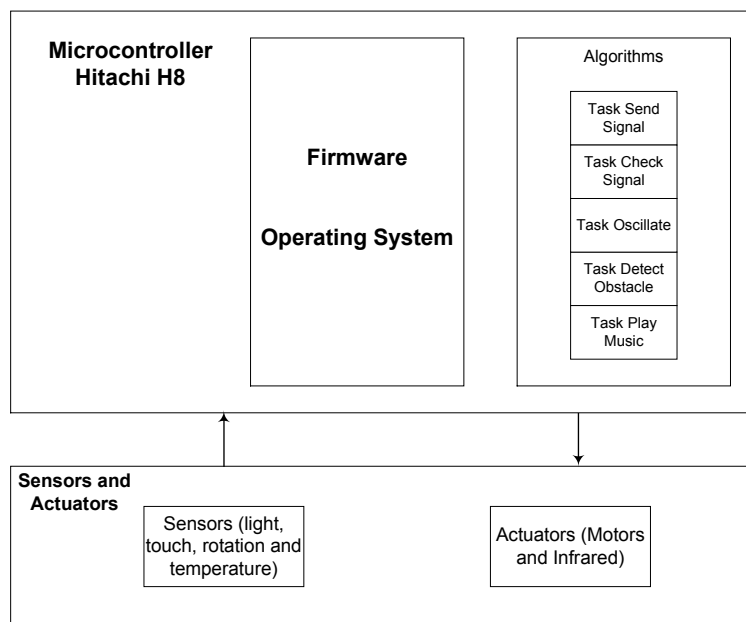


Figure 11 - System Architecture overview

The operating system that will be used (RIS, BrickOS, LeJOS) will provide an API to all the RCX sensors and actuators. The algorithms were written in NQC, C++ and Java. Figure 12 depicts all programming languages and operational systems that will be responsible by the functionality of the system.

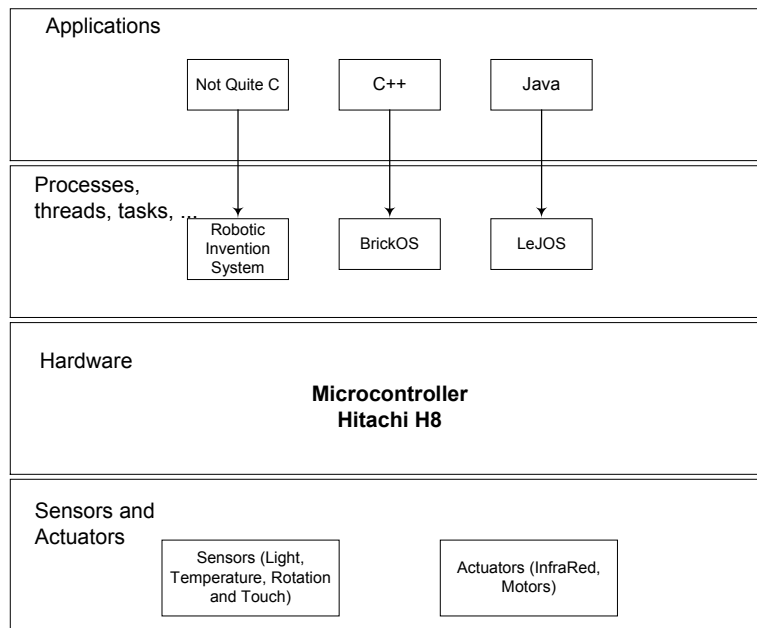


Figure 12 - Operating Systems and Programming Languages

3.3.2 State Diagram

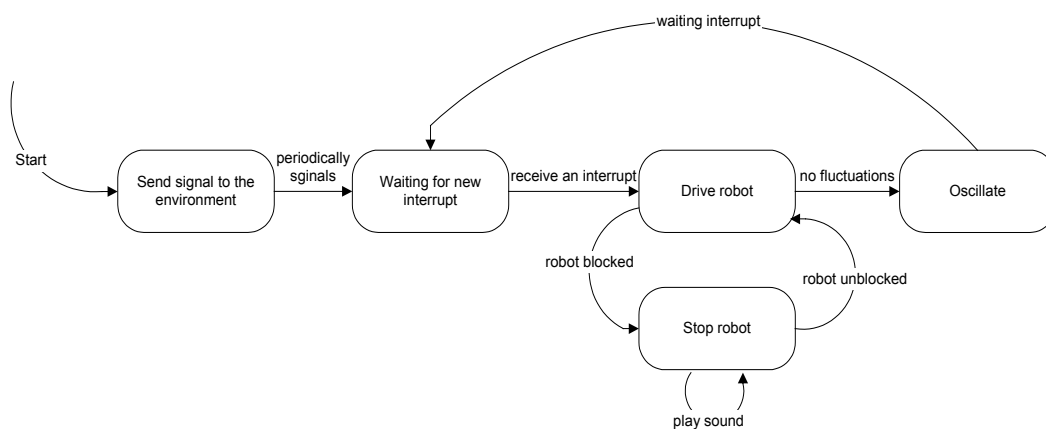


Figure 13 - State Diagram

The correspondence between state diagram and software components are presented in table 2:

Table 2 - Relationship between the state diagram and the software components

States of the diagram	Software components
Send signal to the environment	Send signal
Waiting for new interrupt	Check signal
Drive robot	Check signal
Stop robot	Detect Obstacle
Stop robot	Play sound
Oscillating	Oscillate

3.4 Software Components

3.4.1 Send Signal

The main function of this component is to send signals periodically to the environment. This Send signal component will be written in three different programming languages as presented in the System Architecture document. A light sensor connected in front of the robot will detect the fluctuations as well as possible. The value of the sensor will be displayed on the LCD (Liquid Crystal Display) with the same frequency that the signals are sending to the environment. The three programming languages provide different methods to send data over the infrared and access the LCD, table 3 and 4 depicts the different methods.

Table 3 - Methods to send a signal to the environment

Programming Languages	Send Data	Description
NQC	SendMessage(int msg)	Send data to the receiver. We can to send a value between 0 and 255
BrickOS	Int send_msg(unsigned char msg)	Send a message to the receiver
LeJOS	Boolean sendPacket(byte[] buffer, int offset, int count)	Sends packets to the receiver.

Table 4 - Methods to have access to LCD

Programming Languages	Method	Description
NQC	SelectDisplay(mode)	Select one of 7 display modes: 0: system watch, 1-3: sensor value, 4-6: output setting. Mode may be an expression.
BrickOS	lcd_int(int I)	Display an integer in decimal
	Lcd__unsigned(unsigned int u)	Display an unsigned value in decimal
	Lcd_clock(int t)	Display a clock. Passing an argument of 1015 will display 10.15
	lcd_digit(digit d)	Display a single digit right of the man symbol
LeJOS	LCD.setNumber(int aCode, int aValue, int appoint)	Sets a number to be displayed in the LCD
	LCD.showNumber (int aValue)	Shows an unsigned number on the LCD. The parameter aValue is an unsigned number in [0, 9999]

To achieve a given frequency in which the signals are sent to the environment, a delay statement is necessary and the programming languages provide the following methods depicted in table 5.

Table 5 - Delay Methods

Programming Languages	Method	Description
NQC	Wait(int time)	Wait for the specified amount of time in 100ths of a second. Time may be an expression
BrickOS	sleep(long aSeconds) and msleep(long aMilliseconds)	Wait for the specified amount of time. The sleep statement in seconds and msleep statement 100ths of a second. Time may be an expression
LeJOS	Thread.sleep(long aMilliseconds)	Wait for the specified amount of time in 100ths of a second. Time may be an expression

The RCX is a typical example of real-time system. A real-time clock is necessary because the input sensor signal must be sampled in a regular interval. Thus, the RCX has to incorporate time in its actions in two scales:

- **Macroscopic real-time aspects:**

1. The duration of motors-on signals and the *positioning* of the robot are directly interrelated;
2. Most meaningful robotic *strategies* have to make use of delays and time-outs at some point;
3. The *reaction time* has to be bounded – for example, motors must stop quickly after an obstacle is hit;
4. The *sampling time* of sensors has to be short enough – for example, if a light sensor is used to detect whether a black line is crossed, the necessary sampling rate depends on the environment (line width) and on the robot (speed).

- **Microscopic real-time aspects:**

1. The motors are controlled using *pulse-width-modulation*.
2. The *speaker* must be able to generate various frequencies.

The RCX is controlled by a digital CPU, but still quite different from a non-embedded computer. The table 6 and 7 depicts the contrasts.

Table 6 - The RCX is a typical embedded system

	RCX	Laptop
Applications	Robot Control	“General purpose” Office applications, SW development and etc.
Cost	€ 220,00 Includes SW and building material for a robot	€ 1.300,00 Barely any SW, no peripherals.
Size	$6.5 \times 3.5 \times 9.5 = 216 \text{ cm}^3$	$31.0 \times 3.5 \times 26.5 = 2875 \text{ cm}^3$
Weight	220 g	2850 g
Power	6 AA batteries	16V, 4.5A (72W) transformer
SW Updates	To be avoided	No problem

Table 7 - RCX VS. LAPTOP – Hardware Comparisons

	RCX	Laptop
CPU	Hitachi H8 (8-Bit microcontroller)	Pentium III (32-Bit microprocessor)
Speed	16 MHZ	800MHZ
RAM	32 KB	192 MB
ROM	16 KB	192KB
Addl. Storage	None	20 GB hard drive
Display	1 x 3 cm ² LCD	$28.5 \times 21.5 \text{ cm}^2 = 613 \text{ cm}^2$ TFT
Keyboard	4 buttons	94 buttons + mouse
Further	3 sensor, 3 actuators	USB, serial, parallel, Ethernet, modem
Interface	IR port, speaker	PCMCIA, int./ext. speakers, int./ext. mike, ext. monitor, ext. keyboard/mouse, CD/DVD, floppy, IR Port

As depicted in table 6 and 7, different requirement deal differences in design. The RCX is a typical example of embedded system and must by definition respond to real-world events. In the real world, and in the systems controlled by embedded system, things happen concurrently: in a car, the four wheels spins and the motor runs, all at the same time. Consequently, the fundamental notion in embedded system is concurrent behavior of the software, and of the system that the software controls.

The figure 14 depicts a schematic of a real-time computing. The software which controls the operations of the system can be written in modules which contain the algorithms necessary for physically controlling the devices and a module to retrieve and display the sensors and actuators changes.

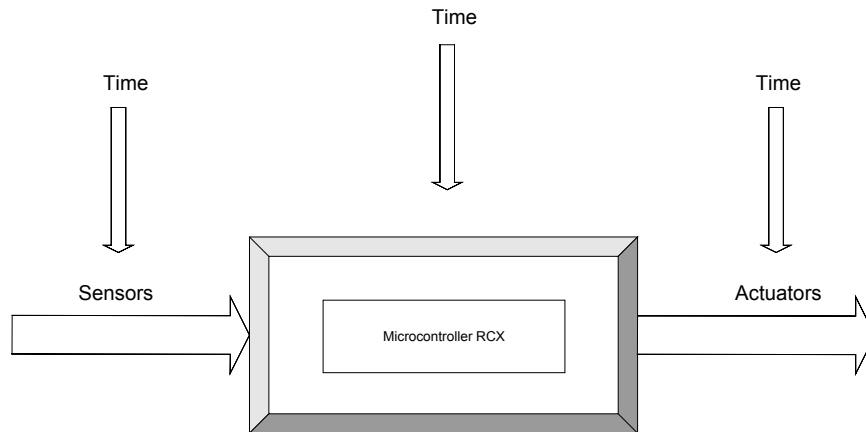


Figure 14 - Real-time computing

Our real-time application must take in account the real-time features provided by the programming languages and operating systems. Different programming languages and operating systems provide different real-time features and table 8 depicts the following characteristics:

Table 8 - Real-time features

Priority	Real-time features	NQC	BrickOS	LeJOS
1	Timeliness	++	++	++
2	Responsiveness	++	++	++
3	Correctness	+	++	++
4	Requirements for Concurrency	++	++	++
5	Sharing Resources	0	++	++
6	Predictability	+	++	+

++ → Very good; + → Good and 0 → Absent

NQC stands for *Not Quite C* and is a programming environment developed by Dave Baum [5]. NQC compiles the same byte code that the standard RIS language produces. The timely compilation of input data, data processing and delivery of output data can be achieved in NQC (timeliness). NQC suffers from some of the limitations of the standard RIS firmware: there is only integer arithmetic (the firmware lacks floating point). There are many applications that do not require floating point, but depend only on the logical result of the computation (correctness) and the time at which the results are produced (timeliness). All variables in NQC are 16-bit integer, and the amount is severely limited: the RCX firmware allows up to 32 global variables. NQC provides classes and methods to interact with the sensors and actuators (responsiveness).

The NQC language includes inline functions⁹, including arguments, but those functions cannot have return values. NQC also allows subroutines – which are not inline, thus

⁹ One means of reducing the overhead is to substitute the code for the subprogram “inline” whenever a call of that subprogram is made and allows the programmer to use subprograms but not incur the run-time overhead.

saving scarce RAM – but subroutines cannot take parameters and cannot call other subroutines. The RCX is limited to 8 subroutines. NQC extends C in adding “tasks”, which run as concurrent threads in the RCX hardware (requirements for concurrency), but it supports up to 10 concurrent tasks. The table 8 summarizes the real-time features described above and assigns a priority to each one in compliance with the application developed. NQC is free software available under the Mozilla public license, and runs under Mac OS and Windows.

BrickOS created by Markus [6] is a multi-tasking operating system running on the RCX hardware and providing an API to all the RCX sensors and actuators. BrickOS comes with firmware that replaces the original RIS firmware and provides a library of system calls for use in a C++ program for the RCX. The timely compilation of input data, data processing and delivery of output data can be achieved (timeliness). The interaction with the sensors and actuators make capable to develop control system use environmental feedback to ensure that the control loop is acting properly (responsiveness). BrickOS provides only the basic Portable Operating System Interface (POSIX) functionality: process management, event notification and counting semaphore. BrickOS provides mechanisms by which tasks are created (requirements for concurrency) and can be assigned priority numbers to tasks (fixed priority).

Another problem in concurrent systems is the robust sharing of resources. BrickOS provides the use of semaphore to serialize the direct access to the resources (Sharing Resources). The more common aspect of predictability is with respect to schedulability and memory management. Embedded systems usually do not provide sufficient memory to the user program. The micro-controllers that are used in embedded applications have the amount of memory in the range of KB and the RCX microcontroller is a good example. The operators *malloc*¹⁰ and *free* provide means to execute the dynamic allocation of memory but especially in embedded real-time applications, the dynamic allocation represents a negative point because the availability of memory as described above. For example, the operator *malloc* raise an exception if there is no available space to create an object. Creating objects on the stack is the most efficient way to allocate storage for objects and to free that storage. Creating objects on the heap can be much more expensive. BrickOS provides means to implement scheduling methods and create objects on the heap. BrickOS is free software available under the Mozilla public license, and runs under Linux, Unix and Windows.

LeJOS software created by Solorzano [7] is an implementation of part of the Java Virtual Machine (JVM) running on the RCX hardware. LeJOS comes with firmware that replaces the original RIS firmware; the new firmware implements many core Java classes,

¹⁰ C’s standard library routine for storage allocation. It takes the number of bytes required and returns a pointer to a block of that size. Storage is allocated from a heap, which lies after the end of the program and data areas. Memory allocated with *malloc* must be freed explicitly using the *free* routine before it can be re-used.

including integer and floating arithmetic, memory allocation, strings, threads, and exceptions. The timely compilation of input data, data processing and delivery of output data can be achieved (timeliness). LeJOS provides classes and methods to interact with the sensors and actuators (responsiveness). LeJOS provides the mechanisms by which threads are created (requirements for concurrency) and can be assigned priority numbers to tasks (fixed priority).

Hence any class that wishes to express concurrent execution must implement the predefined class `java.lang.Thread` and provide the `run` method. LeJOS is free software that runs under Linux, Unix and Windows.

Our application requires only the following real-time features: timeliness, responsiveness, correctness, concurrency and sharing resources. The application must react to the inputs signals from the system's environment and deliver output signals to influence in the environment. Several computation processes must be executed simultaneously.

Figure 15 depicts a schematic of the robot using the infrared port to send signals periodically to the environment.

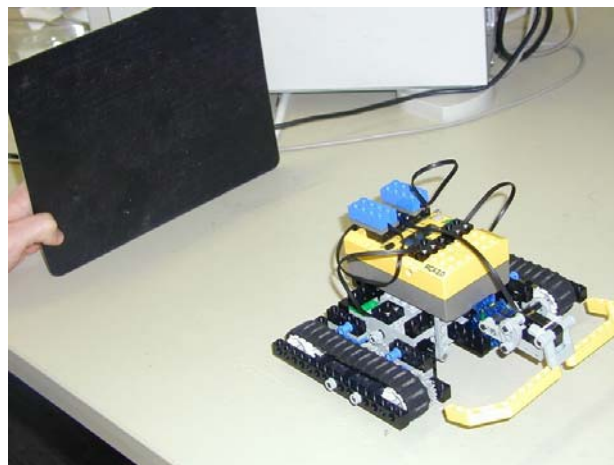


Figure 15 - The robot using the infrared and the light sensor

3.4.2 Check Signal

The “Check signal” component is responsible to drive the robot. A unit with reflection properties will cause the fluctuations detected by the light sensor. The light sensor has a red LED and a light-sensitive diode that responds to infrared light. The light sensors usually are used to read the difference between light and dark areas. For our purpose, this can be used to detect dark object against a light background. An initial value is stored in a variable to work as a base distance between the robot and the unit with reflection properties.

The robot will move with two motors, each one connected in one wheel, for this purpose the programming languages provide different methods to control the motors speed, table 9 and 10 depicts such contrasts.

Table 9 - Motors Control

Programming Languages	Set power	Set direction/turn on the motors	Turn off the motors
NQC	SetPower('motors', 'power');	OnFwd('motors'); OnRev('motors');	Off('motors');
BrickOS	Motor_a_speed(unsigned char speed); motor_b_speed(unsigned char speed); motor_c_speed(unsigned char speed);	motor_a_dir(MotorDirection dir); motor_b_dir(MotorDirection dir); motor_c_dir(MotorDirection dir);	motor_a_dir(off); motor_b_dir(off); motor_c_dir(off);
LeJOS	Motor.A.setPower(int aPower) Motor.B.setPower(int aPower) Motor.C.setPower(int aPower)	Motor.A.forward(); Motor.B.forward(); Motor.C.forward(); Motor.A.reverse(); Motor.B.reverse(); Motor.C.reverse();	Motor.A.stop(); Motor.B.stop(); Motor.C.stop();

Table 10 - Comments regarding the motors control

Programming Languages	Advantage	Disadvantage
NQC	Allows high level coding – for example “motor off” instead of “increment X” (decrease space requirements)	Provide methods to deal with 8 different power levels (0 = no power = off and 7 = full power)
BrickOS	Provide a fine grained control of the motors speed	No disadvantage. Provide method to deal with 256 different power levels (0 = no power = off and 255 = full power)
LeJOS	The public class motor work like the NQC in terms of motors speed	The disadvantage is the same presented to NQC

In reality using the power level make big differences, especially when the motors are not heavy loaded. The programming languages provide different ways to work with the light sensor, table 11 depicts such contrasts. Figure 15 depicts the use of the light sensor and the motors in our application.

Table 11 - Modes to operate with the light sensor

Programming Languages	Light sensor mode
NQC	There are two modes raw and percent
BrickOS	There are two modes active and passive
LeJOS	There are two modes raw and percent

In front of the light sensor brick, there are two components in the brick. The first is the actual light detector, and the second is a small light source. The idea is that the light is turned on whether we desired to find something reasonably close, which will have a big difference in reflectivity from the surrounding. This will amplify the difference between the light and dark (much like shining a flashlight on something). Our application works with this mode. If we desired to judge the environment (finding a white spot on the wall a distance away) the built in light may interfere.

As depicted in table 11, NQC and LeJOS work with the same mode. Percent mode is the most useful mode for light sensors and we will have values between 0 and 100. Raw mode works, but jumps around an average value, in this case we will have values between 0 and 1024.

BrickOS work with active and passive mode. In active mode, the light sensor currently returns values between roughly 50 and 300. However, this may be fixed soon, so that light scales more linearly between 0 and 100. In passive mode, the sensor itself is powered, and different values (between 220 and 280) will occur. They are also not as stable as when the sensor is in active mode. It is quite possible that our range will be different than this for any number of reasons: one of them is when the battery is low.

3.4.3 Oscillate

The “Oscillate” component is responsible to oscillate the robot to back and forth. When the light sensor value is larger than the base distance the robot will drive backwards until no fluctuations have been detected and then will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation. When the light sensor value is smaller than the base distance the robot will drive forwards until no fluctuations have been detected and then will oscillate back and forth as a simple harmonic motion with a period T for one complete oscillation.

3.4.4 Detect Obstacle

The “Detect obstacles” component is responsible to block the robot. Whether there is an obstacle behind the robot, the touch sensor connected behind it will stop the execution of the processes. When the touch sensor is released the robot keep going its normal execution.

The touch sensors are basically switches, they are open (sensor released) or closed (sensor pressed). The programming languages provide different modes to operate with the touch sensor, table 12 depicts the differences.

Table 12 - Modes to operate with the touch sensor

Programming Languages	Touch sensor mode
NQC	There are five modes: Boolean, Raw, Percent, Pulse, Edge mode
BrickOS	There is only the Boolean mode
LeJOS	There are five modes: Boolean, Raw, Percent, Pulse, Edge mode

As depicted in table 12, NQC and LeJOS provide the same modes to work with the touch sensor and BrickOS provide only one mode. Table 13 depicts the five different modes.

Table 13 - Touch sensor modes description

Mode	Description
Boolean	Values 0 and 1 correspond to released and pressed state.
Raw	Values are above 1000, when the sensor is released, and below 200, when it is pressed.
Percent	We will see values between 0 (released) and 100 (pressed), in contrast to raw mode, percent mode gives a stable reading and does not depend on the specific sensor exemplar.
Pulse	It is based on Boolean mode and counts the transitions from 1 to 0. This corresponds to touch sensor releases.
Edge	Almost the same as pulse mode but counts all transitions from 0 to 1 and back.

In our application we will use the Boolean mode, because it is the most useful mode for touch sensor. Pulse mode makes sense for touch sensor but the pulse mode (as well as edge mode and angle mode) returns an accumulated value, not the current state of a sensor.

3.4.5 Play Sound

The “Play sound” component is responsible by sounds generation. When the touch sensor is pushed the robot will generate sounds until the obstacle be removed. When the obstacle has been removed the processes will be released again.

The programming languages provide basically the same modes to operate with the sound, they play a sound with a given frequency and duration, table 14 depicts the methods used to operate the RCX sound.

Table 14 - Sound Methods

Programming Languages	Method
NQC	PlaySound()
BrickOS	Sound::beep()
LeJOS	Sound.beep()

The methods depicted in table 14, play a sound with a given frequency and duration. Frequency is audible from about 31 Hz to above 10000KHz. The duration argument is in $\frac{1}{100}$ of a second. To achieve such frequency a delay statement is necessary and table 5 described in subchapter 3.4.1 depict the property of each method.

3.5 Test Specification

3.5.1 Introduction

Multiple tests had to be realized during the whole implementation phase and also at the end of the integration phase, to analyse whether the system requirements were kept. In the next subchapters we give more details regarding the execution of these tests.

3.5.2 Test Requirements

The test of the software components described in System Architecture and Components Specification require the following elements:

- 1 Unit with reflection properties.
- 2 Light Sensors.
- 1 Touch Sensor.
- The Lego robot with the batteries properly charged.
- The software must be previously installed on the RCX microcontroller.

The real-time application was implemented in three different programming languages and the performance provided for each one was compared. The running functionality of the monitoring application was also checked. It was done displaying the exact monitored value of the light sensors connected in front of the robot.

Additional test requirements are:

- The test was realized in the IAS, within different environment because we had to realize the test with different light flows.

All functionality provided by the software was tested.

3.5.3 Test Methods

To test the functionality of the software system, the Black box test method is employed, i.e., the test object is considered as a black box. The person who tests the application is not interested in the implementation details. He or she is only interested in the inputs and outputs of the system.

The robot will be placed in a position and the unit with reflection properties will be placed in a certain distance of the robot. When the software is initiated, the initial value of the light sensor is stored in a variable and the robot will have a base distance to control the fluctuations caused by the displacement of the unit with reflection properties.

During these the following information must be considered:

- Our real-time application is not interested in how much the robot has moved, we are interested in the difference between the base distance and the displacement detected by the light sensor, only this variation and not how many centimeters or meters the robot has moved.
- The velocity of the displacement should be taken into account because the light sensors have many limitations related the read values.
- The light sensor usually jumps around an average value and then its oscillation becomes difficult to be fulfilled.
- The robot should detect the presence of collision using the touch sensor connected behind it.

3.5.4 Test Criteria

The test will be considered successful whether the following conditions are fulfilled:

- The robot detects the displacement caused by the obstacle with reflection properties and drive according with its displacement.
- The robot oscillates according the fluctuations detected by the light sensors.
- The robot detects the presence of obstacles behind it.
- The robot generates sounds when an obstacle is detected.

3.5.5 Test Cases

We have to test the following cases to see if our software keeps the system requirements.

- The robot must drive according with the displacement caused by the unit with reflection properties. When the unit with reflection properties is closer than the base distance, the robot must drive backwards. Whether the unit with reflection properties is more distant than the base distance the robot must drive to forwards.
- The robot must oscillate according with the fluctuations detected by the light sensors. After that no fluctuations have been detected by the light sensor, the robot must oscillate back and forth as a simple harmonic motion.
- The touch sensor will detect the presence of an obstacle behind the robot and it will be blocked until no obstacle is detected. When no obstacle has been detected the robot must back to its last execution point.
- When the touch sensor detects an obstacle, the robot must generate sounds with a certain frequency to the environment. When the touch sensor is released the generate sound must be released as well.

3.5.6 Test results

In this subchapter we will shortly describe how were the results of the Test specification from subchapter 3.5.

During the development phase the test cases from subchapter 3.5.5 were considered. We could observe, that all the tests that were planned were successfully realized. Therefore, the software could be completed and integrated in the whole system.

The test showed also that the efficiency of the oscillation is strongly depended of the environment. Light flows not continuous prejudice the oscillation component performance. When the unit with reflection properties was closer than the base distance, the robot drove

backwards and when the unit with reflection properties was more distant than the base distance the robot drove forwards.

The robot has detected the presence of obstacle behind it. When the robot has detected an obstacle it has blocked all process related with the motors and has generated sounds to indicate the presence of an obstacle.

I would like to remark, the light sensor that comes with the Lego Mindstorms kit has many limitations. We have used in our application 2 light sensors to detect the fluctuation as well as possible because sometimes the light sensor value jumps around an average value. Even so, using two light sensors we do not have a good fluctuation detected. For example, when a short displacement is caused the robot oscillate back and forth as simple harmonic motion and after few seconds the robot stop to oscillate.

4 Using the Software

4.1 Installation

4.1.1 Prerequisites

For the operation of the system we will use the operating system Microsoft Windows XP, Home Edition, Version 2002 installed on the PC desktop. NQC is free software developed by Dave Baum [5] and available under the Mozilla public license, and runs under Mac OS and Windows. BrickOS is free software developed by Markus L. Noga's [6] (BrickOS project is in transition from legOS to the new name BrickOS) and available under the Mozilla public license, and runs under Linux, Unix and Windows. LeJOS is free software developed by Jose Solorzano [7] and available under the Mozilla public license and runs under Linux, Unix and Windows.

4.1.2 Accessing the embedded microcontroller of Robot

Before installing any programming environment, you must to have a medium to access the Lego embedded computer. The IR tower is responsible for sending data from PC to microcontroller RCX and receives data from RCX microcontroller to PC. The communication between two RCX is also possible. The Lego Mindstorms basic kit comes with CD-ROM that install the IR Tower and all software to run the Robotic Invention System (RIS). The NQC programming language use the same firmware that comes with LEGO Mindstorms kit and it

needs some packages of this software. You have to install the software that comes with LEGO Mindstorms as well as the IR tower.

When you develop your software in the programming languages you need to send it to the RCX over the infrared port. The distance between the IR Tower and the RCX must be taken into account. Our advice is to set the distance between the RCX and the IR Tower as far as possible. If you have already installed the LEGO USB Tower (the USB tower does not require the use of batteries) the figure 16 shows how could set it property. The data baud rate can be set in the *Advanced* tab control but our advice is to accept the default value.



Figure 16 – LEGO USB Tower

4.1.3 NQC Installation

NQC was developed as a command-line tool, but others programmers have created Integrated Development Environments (IDEs). They provide a nice user interface on top of the standard NQC compiler. They also add features such as remote control of the RCX, syntax highlighting, etc. We have used Bricx Command Center. For Bricx Command Center installation you have to follow these steps:

1. You must download the Bricx Command Center software [5].
2. You have to execute the file *bricxcc_setup_3375*, which you have downloaded.
3. The next step requests your name and company.
4. The location to install the software can be set.
5. In the next step the type of the installation is requested. For this purpose, we have used the typical installation.

6. The last step request a new folder name or selects an existing one from the existing folder list.

When the Brick Command Center has been installed you will see the figure 17 with a project opened by us to show the highlighting.

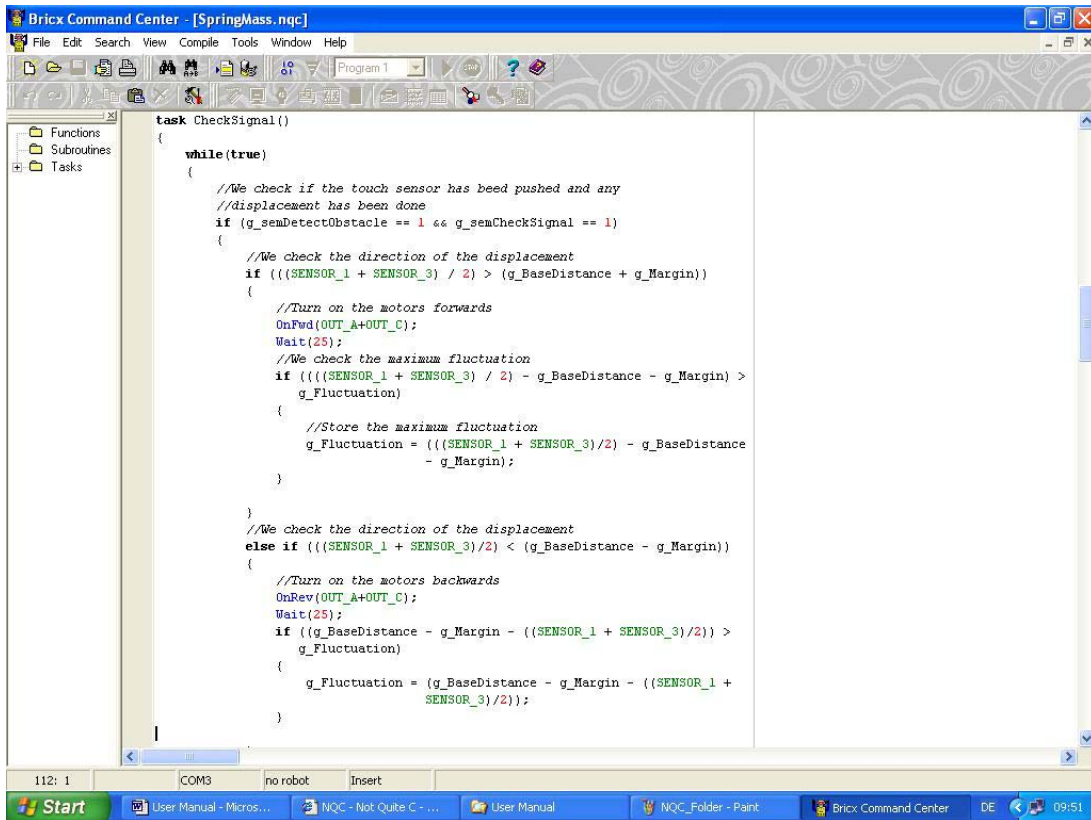


Figure 17 - Brick Command Center Environment

4.1.4 BrickOS Installation

The installation of Cygwin 1.3.x or newer version is necessary to run the BrickOS. We have used the version 1.3.20. Cygwin is a Linux-like environment for Windows. It consists of two parts:

1. A DLL (cygwin1.dll) that acts as a Linux emulation layer providing substantial Linux API functionality.
2. A collection of tools, which provide Linux look and feel.

The Cygwin DLL works with all non-betas, none "release candidate", ix86 versions of Windows since Windows 95, with the exception of Windows CE. There have also been reports of problems on Windows Server 2003.

To install all software to run the BrickOS you have to follow these steps:

1. Download the latest cygwin version from Cygwin site[4].
2. You must choose which modules you need to download and setup. Table 15 present a minimal list (but probably some modules could be removed as well) that was tested and works.
3. Open a cygwin bash shell window (you can open it using *Start->Programs->Cygnus Solution->Cygwin Bash Shell*).
4. Make a new directory with the command line *mkdir /build*.
5. Download gcc 2.95.2 (or a newer version) sources [5] and save in *c:\cygwin\build*.
6. Download binutils 2.10.1 (or a newer version) sources [6] and save in *c:\cygwin\build*.
7. Download the building scripts [7] and put it in the same directory as above. Unzip them to produce a shell script (buildgcc.sh) and two diff files.
8. Build the cross compiler with the two command line, first *cd /build* (enter) and after *./bluidgcc.sh* (enter).
9. Download the brickos-0.2.6.10.tar.gz [2] and save in *c:\cygwin*.
10. Type the two command lines, first *cd/* (enter) and after *tar xvfz brickos-0.2.6.10.tar.gz* (enter).
11. Type the two command lines *cd /BrickOS* (enter) and after *./configure*;
12. Type more two command lines *cd util* (enter) and after *make strip*. Now you PC is set to run the BrickOS program.

Table 15 – Modules to download and install the cygwin

Ash	Autoconf	Automake	Bash	Binutils	Cpio
Cygwin	Diff	File	Fileutils	Findutils	Flex
Gcc	Grep	Less	Login	Make	Mingw
Sed	Shellutils	Tar	Textutils	Time	W32api

4.1.5 LeJOS Installation

In order to install the LeJOS follow these steps:

1. Download LeJOS software [7].
2. Unzip the file you have downloaded. A top directory, lejos, is already provided.
3. Set RCXTTY to the IR USB port, e.g. *set RCXTTY=usb*.
4. Both the JDK's and leJOS' `bin` directories should be in your PATH.
5. Under Windows 95/98, you might want to create a batch file that sets these variables, and specify that batch file in the `Program` properties of a DOS console shortcut; or set them in *C:\AUTOEXEC.BAT*.

We have used the software Eclipse [12] as an Integrated Development Environment to develop our application. The Software Eclipse is free and can be set to leJOS. In order to set the Eclipse and LeJOS follow these steps:

1. Download the plug in org.leJOS_1.0.2.zip[13] and save in c:\eclipse\plugins.
2. In the Eclipse software go to the leJOS preference page, using *Window->Preferences->Lejos page*. The figure 18 shows the window .
3. Make the installation path point to a valid LeJOS installation. You can use the directory dialog or type directly the location.
4. Configure the RCX tower communication port (com1, com2 or USB) and data transfer speed. Fast mode and com1 are the default.
5. You can write your programs in the Eclipse software as well as to download the firmware and the byte code to the RCX microcontroller over the IR Tower. Figure 19 shows the LeJOS menu in the software Eclipse.

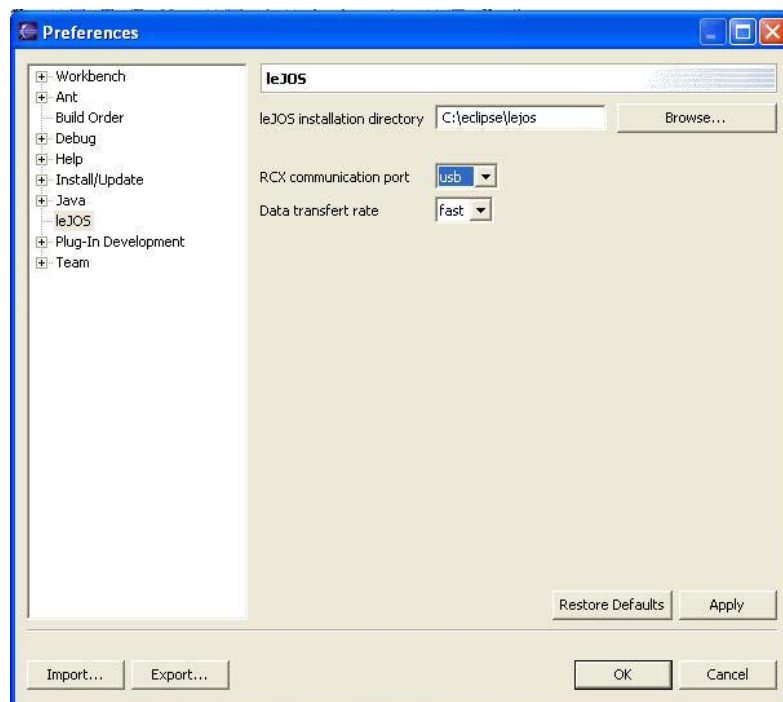


Figure 18 - LeJOS preferences

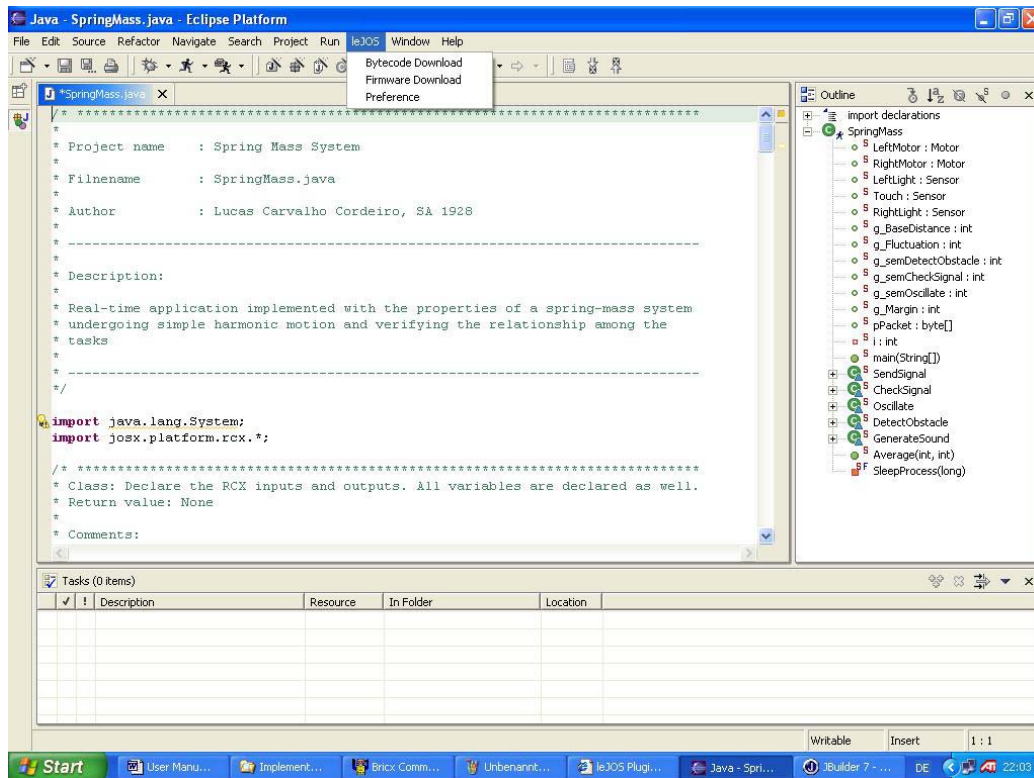


Figure 19 - LeJOS menu

4.2 Operation

Whether you have already successfully installed the needed programs, you can use the robot for our applications in a very easy way.

4.2.1 Preparation of the robot

You have to follow these steps:

1. The Lego robot with the batteries properly charged.
2. The firmware and software must be previously installed on the RCX microcontroller.
3. The robot should be tested within an environment that a light flow is continuous.
4. An obstacle with reflection properties should be used (we have used a mouse black pad).

4.2.2 Running NQC programs

In order to run NQC programs you must first to follow the steps depicted in section 4.1.3. You can write your programs in the Bricx Command Center and download to RCX microcontroller. To do this follow this steps:

1. Click in *Start -> Programs -> Bricx Command Center -> Bricx Command Center*. A window will request the port and firmware that you desired to work. The figure 20 shows this window.
2. In the Bricx Command Center click in *File -> New* to edit new software.
3. After that you have already written your program you can save in a directory that you desired clicking *File -> Save*.
4. Now you have to download the firmware that comes with the Lego Mindstorms kit. Click in *Tool -> Download Firmware* (Whether you are using the RIS 2.0, the name of this file is firm0328.lgo). This process takes about 5 minutes.
5. Click in *Compile -> Compile* or press F5 to compile your software.
6. Click in *Compile -> Download* or press F6 to download your software to RCX microcontroller. You can also download and run the software clicking in *Compile -> Download and Run* or press Ctrl + F5. This process takes about 1 minute depending the size of your program. Now you can run your application.

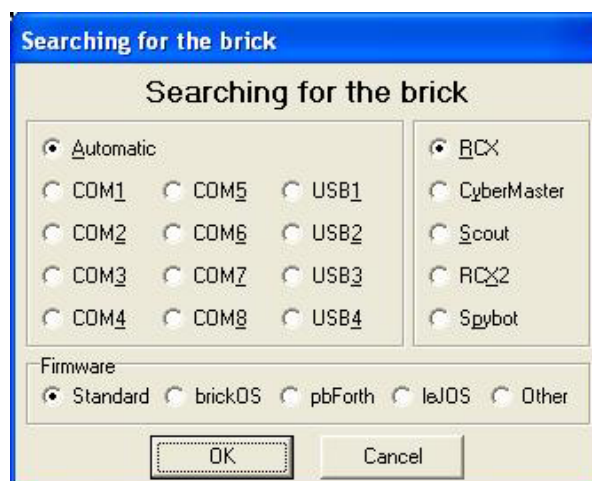


Figure 20 - Searching for the brick

4.2.3 Running BrickOS programs

In order to run BrickOS programs you must first follow the steps depicted in section 4.1.4. We have used the Visual C++ IDE to write our application. To run the BrickOS programs follow these steps:

1. Write your program in Visual C++ or other Integrated Development Environment that support the C++ programming language. Thus, you can use the highlighting code to facilitate the software development.
2. Create a directory inside the *c:\cygwin\brickos-0.2.6.10* and save your programs with the extension *.C.
3. Copy the file *Makefile* from the directory *C:\cygwin\brickos-0.2.6.10\demo* to the directory in which you have saved your program.
4. If there is no firmware stored in the RCX then you must go to the directory *C:\cygwin\brickos-0.2.6.10\util* and download the firmware to the RCX microcomputer with the command line *./firmdl3 ../boot/BrickOS.srec*. This process takes about 5 minutes and it is done only one time, this means that if you turn off the RCX, the firmware will keep on doing stored in the memory.
5. The programs written with the extension *.C must be compiled to produce the files with the extension *.lx. The command *makefile Name_of_the_file.lx* is responsible for this process.
6. Now you can download your program to the RCX with the command line *./dll ../Your_Directory/Name_of_the_file.lx*. This process takes about 30 seconds depending the size of your program. Now you can run your application.

4.2.4 Running LeJOS programs

In order to run LeJOS programs you must first to follow the steps depicted subchapter 4.1.5. We have used the Eclipse Software to write our application. To run the LeJOS programs follow these steps:

1. Open the file *eclipse.exe* from the directory *c:\eclipse*.
2. Click in *File -> New -> Project*.
3. You have to select the leJOS Wizard as shows in figure 21. The wizard looks like the Java wizard: just type a project name, add some library whether you wish. Do not care about

the LeJOS libs (classes.jar, rcxrcxcomm.jar, pcrxcomm.jar), they will be added automatically.

4. Switch to a Java perspective and create new Java files in that perspective. For now, the leJOS menu only appears when editing a Java file. As stated before, the menu allows you to download the firmware and the current file (which must provide a main method). It also provided a quick shortcut to the preference page.
5. Click in *File -> Save* and choose a directory that you desired to save your project.
6. Click in *LeJOS -> Firmware Download* to download the LeJOS firmware. The figure 22 shows this window in progress. This process takes about 5 minutes.
7. Now you can download your program to RCX microcontroller clicking in *LeJOS -> Byte code Download* and figure 23 shows this window in progress. This process takes about 2 minutes depending the size of your program. Now you can run your application.

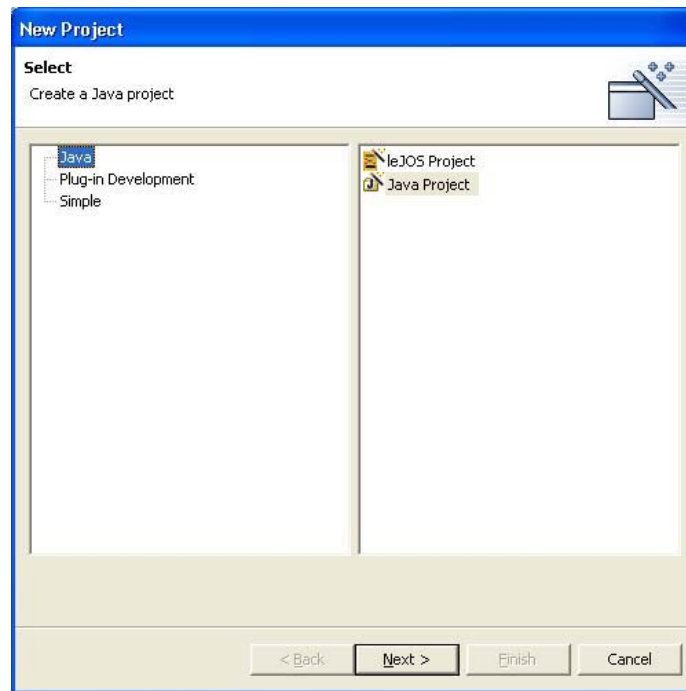


Figure 21 - LeJOS project Wizard

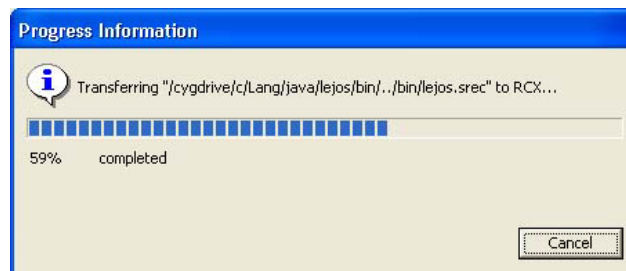


Figure 22 - Firmware Download

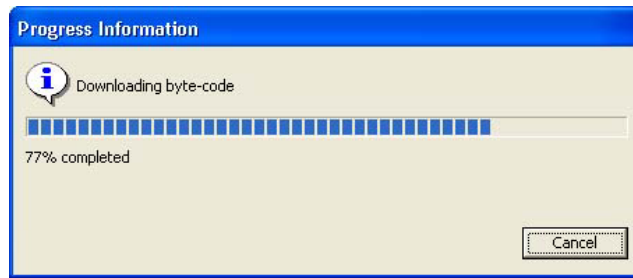


Figure 23 - Byte code Download

4.3 Control Elements

The RCX contains an LCD with a programmable 4-digit section. It can be set to monitor a value (e.g. a timer) continuously with only a single command. The display also shows the current execution state (program running/not running), the fill state of the datalog, the number of the currently selected slot, the state of the motors, and the state of connected touch sensor (small triangles that show up when a sensor is pressed).

The four buttons on the RCX are titled *On/off*, *Run*, *Select*, *Prgm* and *View* as shown in figure 24. The on/off button should be self-explanatory. The Prgm button allows to cycle through five program slots. The program in the currently selected slot (if present) can be started using the Run button. Finally, the View button allows displaying the values of connected sensors in the 4-digit section mentioned above.



Figure 24 - The display and buttons of the RCX

4.3.1 Batteries

This is very important aspect our application. Without the batteries properly charged, the robot will not work correctly or could even no longer work. The Oscillate component is done with delay statement to synchronization of the simple harmonic motion. The motors speed is extremely affected by the flow current supplied by the batteries. Therefore you must to check whether the voltage is about 9V. The batteries also affect the sensors value.

For touch and temperature type input, the RCX has a 10K Ω resistor pulling up the input to 5V. For light and rotation type input, the RCX has a 120 Ω resistor pulling up to 8V (probably a diode drop from the battery voltage) to power the red LED about 3ms and then looks at the sensor voltage during a short 0.1ms time. In other words, whether the batteries are under 8V the light and rotation sensors will not work correctly and under 5V the touch and temperature sensor as well.

4.4 Functions

The purpose of our software is an application with real-time characteristics implemented with different programming languages. The three programming languages provide different ways to interface with the sensor, actuators and LCD. The spring mass system that was developed in this work is an example of event-driven system whose behaviour is primarily caused by specific reactions like the presence of any obstacle or the displacement of the unit with reflection properties.

5 Conclusion and Outlook

5.1 Summary

5.1.1 Introduction

The microelectronic evolution and the CPUs becoming cheaper, smaller, faster and more reliable, so their range of application widens. The main functions in many modern devices are smaller computers that are embedded. These computers are different from ours computers that we have at home, they are designed to perform a main task and it is not necessary to have a hard and floppy disk and keyboard. For instance, household, communication and maintenance devices only become functional using integrated microprocessor-based controls. A microprocessor-controlled wash machine is a good example, which the prime function is to wash clothes, but it depends which clothes will be washed, then there are different “wash programs” that must be executed. These types of computer application are examples of **real-time** and **embedded**.

In compliance with the embedded system, the fields of Robotics have been fruitfully interacting at the research and development level for many years. Unfortunately, until recently two factors combined to seriously limit the use of this programming tool. The first was the high cost of robotics kits (often \$5000 apiece in the early 1990s), and the second was the lack of a framework for comprehensively integrating robotics [27]. We have demonstrated in my

undergraduate project that LEGO Mindstorms has become a suitable platform to develop real-time applications according with its limitations.

Apart from the standard software delivered with the Mindstorms kit, several independent groups have released development environments and operating systems to the RCX microcomputer. Thus, the goal of my undergraduate project was the investigation of the available methodologies, techniques and tools to develop real time applications based on the LEGO Mindstorms kit. In order to illustrate their characteristics, an application scenario with real time characteristics was defined and implemented with the different tools.

5.1.2 Programming languages comparison to the LEGO Mindstorms

The core of my undergraduate project was to analyze the programming languages available to the LEGO Mindstorms taken into account the real-time features. There are many programming languages available to work with LEGO Mindstorms kit and we had to consider many aspects that influence in development of real-time applications and it was not as easy as expected.

There has been a long debate among programmers, language designers and operating system designers as to whether it is appropriate to provide support for concurrency in a language or whether this should be provided by the operating system only. BrickOS provides support to concurrency to program Mindstorms models in the operating system. NQC and LeJOS provide support to concurrency in the language. I have decided to assign the concept very good to the three languages in terms of concurrency. In my opinion, provide support to concurrency in the language can lead to more readable and maintainable programs and usually an embedded computer may not have any resident operating system

5.1.3 An application scenario with real-time characteristics

An application scenario with real-time characteristics was defined and implemented in compliance with the different tools chosen. The spring mass system was chosen as an application scenario to depict the real-time features in the three different programming languages. The physical model of the spring mass system consists of a mass suspended and set to vibrate on a spring. When the mass is hung on the spring and then displaced from its equilibrium position, it will oscillate up and down.

5.2 Experiences

The realization of this work was for me definitively a positive experience. I can fundament this overall appreciation mainly in three aspects:

1. First research experience in Germany: Although I have developed many research and developments projects at Federal University of Amazonas before this work, the change of the culture framework was evident. It was not only a question of language but also a totally new set of non-written rules, references and criterions. This experience will be invaluable in the future development of my master thesis.
2. IAS Quality System: The application of the IAS Process Model during this work proved to be very profitable for the quality of the execution and its results. The IAS Process Model is a suitable method to the structuring of the project with the required flexibility, but at the same time precise enough to ensure an efficient development.
3. Familiarization with real-time system: I have learned a lot regarding the real-time programming, operating systems and software engineering. In this sense I have also learned: formal and structured methods for development, scheduling theories, reuse, language design, kernel design, communications protocols, distributed and parallel architectures and program code analysis. More important than that, I will have a new research line to research and development projects in the area of power system and industrial automation at Federal University of Amazonas.

5.3 Problems

Thanks to the project tutor, Paulo Urbano, has helped me always a lot (even when he had no time due to his tight agenda), and was not only friendly and helpful, but he had also good knowledge to efficiently support my work. I really consider very positive and effective this work and two difficult aspects were constant in the process. They are:

1. Real-time systems fundamental concepts: The lecture real-time programming with Prof. Dr. Erhard Ploedereder and his assistant Prof. Dr. Rainer Koschke was also extremely important to development of this work. I have learned a lot regarding the concepts and design of real-time systems.
2. LEGO Mindstorms kit: Many troubles have come out during the development of this undergraduate project. The LEGO light sensor was the key point of my application because this light sensor that comes with the LEGO Mindstorms kit had not a good

accuracy for my application and I spent a lot of my time trying to solve this problem and the solution found for me was to use two light sensors to have a better accuracy of the values read.

Appendix A Index of Figures

FIGURE 1 - SYSTEM INTERFACE	9
FIGURE 2 - RCX™ MICROCOMPUTER.....	10
FIGURE 3 - SPECIAL MINDSTORMS PARTS.....	13
FIGURE 4 - THE ROBOTICS INVENTION SYSTEM PROGRAMMING ENVIRONMENT	15
FIGURE 5 – TEST PANEL SHOWS THE VALUE MEASURED OF LIGHT SENSOR	16
FIGURE 6 - THE LEJOS JVM MEMORY MAP	19
FIGURE 7 – A TYPICAL EMBEDDED SYSTEM.....	20
FIGURE 8 - A SIMPLIFIED DIAGRAM	26
FIGURE 9 - HOOKE’S LAW	28
FIGURE 10 - SPRING-MASS SYSTEM WITH LEGO MINDSTORMS	29
FIGURE 11 - SYSTEM ARCHITECTURE OVERVIEW.....	30
FIGURE 12 - OPERATING SYSTEMS AND PROGRAMMING LANGUAGES.....	31
FIGURE 13 - STATE DIAGRAM.....	31
FIGURE 14 - REAL-TIME COMPUTING.....	35
FIGURE 15 - THE ROBOT USING THE INFRARED AND THE LIGHT SENSOR	37
FIGURE 16 – LEGO USB TOWER.....	45
FIGURE 17 - BRICX COMMAND CENTER ENVIRONMENT.....	46
FIGURE 18 - LEJOS PREFERENCES	48
FIGURE 19 - LEJOS MENU	49
FIGURE 20 - SEARCHING FOR THE BRICK	50
FIGURE 21 - LEJOS PROJECT WIZARD.....	52
FIGURE 22 - FIRMWARE DOWNLOAD	52
FIGURE 23 - BYTE CODE DOWNLOAD	53
FIGURE 24 - THE DISPLAY AND BUTTONS OF THE RCX	53

Appendix B Index of Tables

TABLE 1 - LEGO MINDSTORMS ROBOTIC INVENTION SYSTEM (RIS) 2.0 COSTS. 14

TABLE 2 - RELATIONSHIP BETWEEN THE STATE DIAGRAM AND THE SOFTWARE COMPONENTS 31

TABLE 3 - METHODS TO SEND A SIGNAL TO THE ENVIRONMENT..... 32

TABLE 4 - METHODS TO HAVE ACCESS TO LCD 32

TABLE 5 - DELAY METHODS..... 33

TABLE 6 - THE RCX IS A TYPICAL EMBEDDED SYSTEM 34

TABLE 7 - RCX VS. LAPTOP – HARDWARE COMPARISONS..... 34

TABLE 8 - REAL-TIME FEATURES 35

TABLE 9 - MOTORS CONTROL 38

TABLE 10 - COMMENTS REGARDING THE MOTORS CONTROL 38

TABLE 11 - MODES TO OPERATE WITH THE LIGHT SENSOR..... 39

TABLE 12 - MODES TO OPERATE WITH THE TOUCH SENSOR..... 40

TABLE 13 - TOUCH SENSOR MODES DESCRIPTION..... 40

TABLE 14 - SOUND METHODS..... 41

Appendix C Abbreviations

IAS	Institut für Automatisierungs- und Softwaretechnik (Institute of Industrial Automation and Software Engineering.
ISTE	Institut für Softwaretechnologie
AS	StudienArbeit
RCX	Robotic Command Explorer
NQC	Not Quite C
BrickOS	Brick Operating System
IR	InfraRed
LASM	LEGO Assembly
OS	Operating System
POSIX	Portable Operating System Interface
ADC	Analog to Digital Converter

Appendix D Terminology

<p>NQC</p>	<p>NQC stands for Not Quite C, and is a simple language for programming several LEGO MINDSTORMS products. Some of the NQC features depend on which MINDSTORMS product you are using. This product is referred to as the <i>target</i> for NQC. Presently, NQC supports five different targets: RCX, RCX2 (an RCX running 2.0 firmware), CyberMaster, Scout, and Spybotics.</p>
<p>BrickOS</p>	<p>BrickOS is an alternative operating system for the Lego Mindstorms RCX Controller. It also provides a C/C++ development environment for RCX programs using gcc and g++ (the GNU C and C++ cross compilation tool chain) and the necessary tools to download programs to the RCX, RCX2 (an RCX running 2.0 firmware), CyberMaster, Scout, and Spybotics.</p>
<p>LeJOS</p>	<p>LeJOS is an implementation of part of the Java Virtual Machine (JVM), running on the RCX microcomputer. It also provides Java development environment for RCX programs and the necessary tools to download programs to the RCX, RCX2 (an RCX running 2.0 firmware), CyberMaster, Scout, and Spybotics.</p>
<p>Bricx Command Center</p>	<p>Bricx Command Center (BricxCC) is a Windows (95, 98, ME, NT, W2K, XP) program commonly known as an integrated development environment (IDE) for programming the RCX (all versions), Scout, Cybermaster, and Spybot programmable bricks using Dave Baum's Not Quite C (NQC) language. It also supports programming the Scout, RCX2, and Spybot using the LEGO Company's MindScript(tm) and LASM(tm) languages via the Mindstorms 2.5 SDK. It additionally supports programming RCX bricks using the brickOS alternate firmware in C, C++, and Pascal. Version 3.3 of BricxCC is an enhanced revision to Mark Overmars' original program.</p>
<p>Eclipse</p>	<p>The Eclipse Platform is designed for building integrated development environments (IDEs) that can be used to create applications as diverse as web sites, embedded Java™ programs, C++ programs, and Enterprise JavaBeans™. Eclipse also has a plug-in that allows the programmer to develop java code for the LEGO RCX brick using the leJOS JVM. It offers easy configuration through a project wizard and a preference page, uses eclipse code building and adds RCX specific operations both for firmware and byte-code download.</p>
<p>RCX Microcomputer</p>	<p>The RCX microcomputer has a piezoelectric speaker, which produces 6 distinct tones. Using infrared communication the RCX can communicate with a computer, sending messages back and forth;</p>

	communicate with other RCX bricks: messages can be passed from RCX to RCX; be controlled via the LEGO Mindstorms Remote Control.
IR Tower	Download the RCX programs from the computer to the RCX via the Infrared Transmitter. Transmitter work on both Mac and Windows platforms. The IR Tower is available in two cable versions: USB cable and Serial cable.
Unit with reflections properties	The unit with reflection properties that allows causing a displacement of robot to back and forth.
Simple Harmonic Motion	Type of vibratory motion in which acceleration of body is directly proportional its displacement and the acceleration is always directed towards the equilibrium (mean) position is called Simple Harmonic Motion.
Spring Mass System	Spring mass system consists of a mass suspended and set o vibrating on a spring. When the mass is hung on the spring and then displaced from its equilibrium position, it will oscillate up and down.

Appendix E Literature

[1]	The LEGO Group. <i>RCX 2.0 BETA SDK</i> , [Online]. Available at http://mindstorms.lego.com/sdk2/index.html [12 th July 2003]
[2]	Mark Overmars (1999). <i>Programming Lego Robots using NQC.</i> , [Online]. Utrecht University, Department of Computer Science. Available at http://www.cs.uu.nl/people/markov/lego/tutorial.pdf [10 th June 2003]
[3]	Kekoa Proudfoot, <i>RCX Internals</i> , [Online]. Available at http://graphics.stanford.edu/~kekoa/rcx/ [20 th July 2003]
[4]	The LEGO Group, <i>LEGO MINDSTORMS</i> , [Online]. Available at http://mindstorms.lego.com/ [6 th June 2003]
[5]	D. Baum, <i>Not Quite C (NQC)</i> , [Online]. Available at http://www.baumfamily.org/nqc/ [12 th June 2003]
[6]	L. N. Markus, <i>BrickOS</i> , [Online]. Available at http://brickos.sourceforge.net/ [25 th July 2003]
[7]	J. Solorzano, <i>LeJOS</i> , [Online]. Available at http://lejos.sourceforge.net/ [28 th July 2003]
[8]	Cygwin , [Online]. Available at http://www.cygwin.com/ [25 th July 2003]
[9]	Gcc 2.95.2 Download, [Online]. Available at ftp://ftp.gnu.org/pub/gnu/gcc/gcc-2.95.2.tar.gz [25 th July 2003]
[10]	Binutils 2.10.1 Download, [Online]. Available at ftp://ftp.gnu.org/pub/gnu/binutils/binutils-2.10.1.tar.gz [25 th July 2003]
[11]	Legos-Buildgcc.zip Download, [Online]. Available at http://legos.sourceforge.net/cygwin/download/legos-buildgcc.zip [25 th July 2003]
[12]	Eclipse , [Online]. Available at http://www.eclipse.org/ [23 rd August 2003]
[13]	org.leJOS_1.0.2.zip Download, [Online]. Available at http://www.info.ucl.ac.be/people/chp/projects/javarcx/eclipse/org.lejos_1.0.2.zip [23 rd August 2003]
[14]	Burns, A. & Wellings A.(2001). <i>Real-Time Systems and Programming Languages</i> . Harlow: Addison-Wesley.

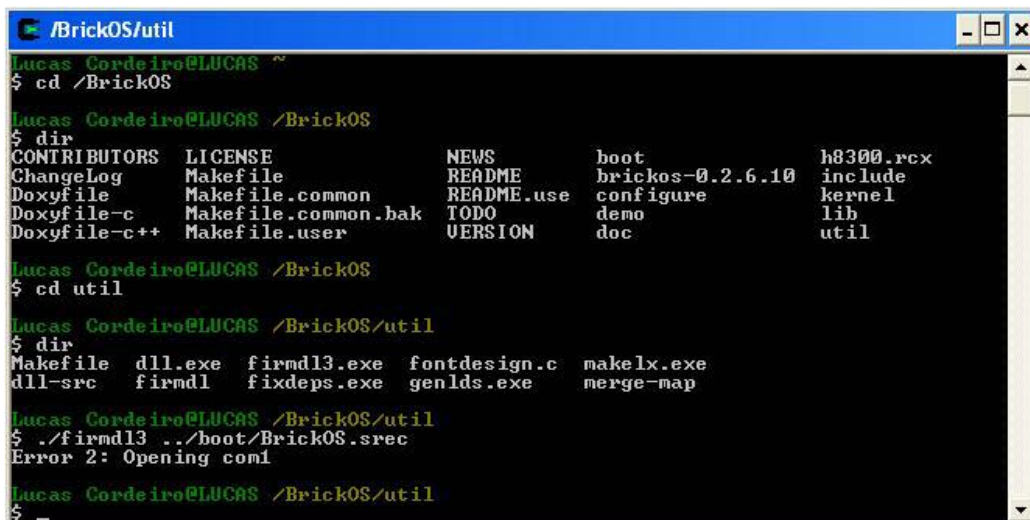
	<i>Languages</i> . Harlow: Addison-Wesley.
[15]	Douglass, B. P. (1999) . <i>Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns</i> . Harlow: Addison-Wesley.
[16]	Harbour, M.G., Klein, M.H., and Lehoczky J.P. (1991) . <i>Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority</i> . Proceeding of the IEEE Real-Time Systems Symposium, IEEE Computer Society Press.
[17]	Göhner, Peter (2003) : <i>Lecture notes of Industrial Automation</i> . IAS, Stuttgart.
[18]	Nissanke, N. (1997) . <i>Real-time Systems</i> , Great Britain, Prentice Hall Europe.
[19]	Barnes, J. (2002) . <i>Programming in Ada 95</i> . Harlow: Addison-Wesley.
[20]	Kopetz, Hermann (2000) . <i>Software Engineering for Real-Time: A Roadmap</i> , International Conference on Software Engineering, Limerick, Ireland.
[21]	Parks, J. E. (2000) . <i>Hooke's Law</i> . Department of Physics and Astronomy, The University of Tennessee, USA.
[22]	Plödereder, E (2003) . <i>Lecture notes of Real-Time Programming</i> . ISTE, Stuttgart.
[23]	Deitel H.M, Deitel P. J (2001) . <i>C++ Como Programar</i> . Bookman.
[24]	Deitel H.M, Deitel P.J. (2003) . <i>Java Como Programar</i> . Bookman.
[25]	Sommerville, I. (2001) . <i>Software Engineering</i> . Harlow: Addison-Wesley.
[26]	Junghans, A. (2001) . <i>Collaborative Robotics with LEGO Mindstorms</i> . Master Thesis, Department of Computer Science, Karlsruhe University of Applied Sciences, Germany.
[27]	Klassner, F., and Anderson, S. (2003) . <i>LEGO MindStorms: Not Just for K-12 Anymore</i> . IEEE Robotics and Automation Magazine.

Appendix F Troubleshooting

In this Appendix we present some problems that occurred during the installation or operation of the software, and possible solutions to them proposed. This Appendix does not intend to be a complete guide regarding the different error situations that we could have, but only an extra help for common errors that I had during the development of the software.

F.1 I do not have access the embedded microcontroller over the IR Tower

The brickOS need to set manually the IR Tower to the port that you desired work. The command to set this variable is `export RCXTTY=port` in the shell window. Where `port` can be set as COMx and USB. The error figure 1 depicts such situation.



```
Lucas Cordeiro@LUCAS ~
$ cd /BrickOS

Lucas Cordeiro@LUCAS /BrickOS
$ dir
CONTRIBUTORS  LICENSE          NEWS             boot             h8300.rcx
ChangeLog     Makefile         README          brickos-0.2.6.10 include
Doxyfile      Makefile.common README.use      configure       kernel
Doxyfile-c    Makefile.common.bak  TODO           demo            lib
Doxyfile-c++  Makefile.user     VERSION        doc             util

Lucas Cordeiro@LUCAS /BrickOS
$ cd util

Lucas Cordeiro@LUCAS /BrickOS/util
$ dir
Makefile  dll.exe  firmdl3.exe  fontdesign.c  makelx.exe
dll-src   firmdl  fixdeps.exe  genlds.exe   merge-map

Lucas Cordeiro@LUCAS /BrickOS/util
$ ./firmdl3 ../boot/BrickOS.srec
Error 2: Opening com1

Lucas Cordeiro@LUCAS /BrickOS/util
$
```

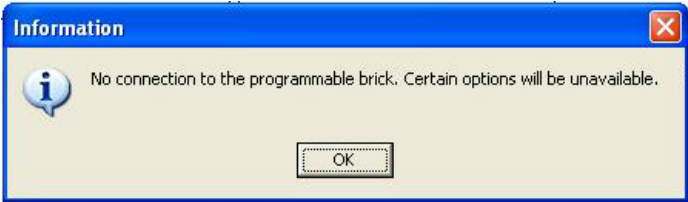
Error figure 1 – Error to open the IR port

The NQC and LeJOS programming language do not have this problem because we have worked with an IDE that has an interface with the user to choose the desired communication port. The figure 20 in subchapter 4.2.2 depicts the port configuration for Brick Command Center. The figure 18 in subchapter 4.1.5 depicts the port communication for Eclipse.

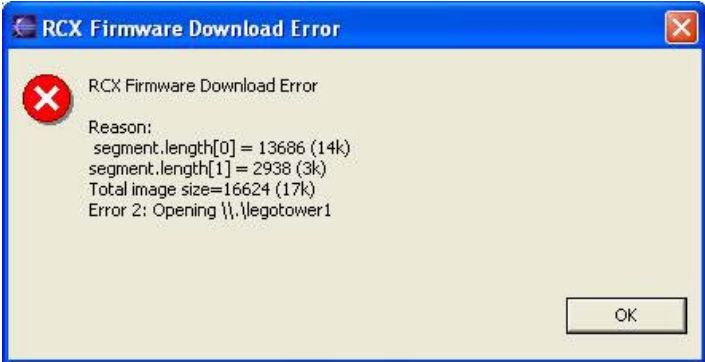
F.2 The IR Tower still does not work

You need to check the configuration of the communication port as described in subchapter 4.1.2. Whether the trouble keep going, try to change the IR Tower to another USB

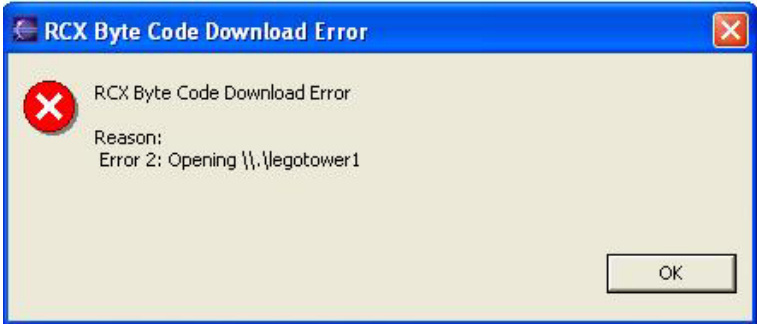
port in your computer. The BrickOS presents the same error as presented in Error figure 1 but with the message *Error 2: Opening legotower1*. The Bricx Command Center shows an error message as depicted in Error figure 2 when do not establish a connection with the RCX microcomputer over the IR Tower. The Eclipse shows an error message as depicted in Error figure 3 when do not download the firmware to the RCX over the IR Tower and Error figure 4 depict the same situation to download the byte code to the RCX.



Error Figure 2 – An error message trying to have access over the IR Tower in NQC



Error Figure 3 – An error message trying to download the firmware



Error Figure 4 – An error message trying to download the byte code

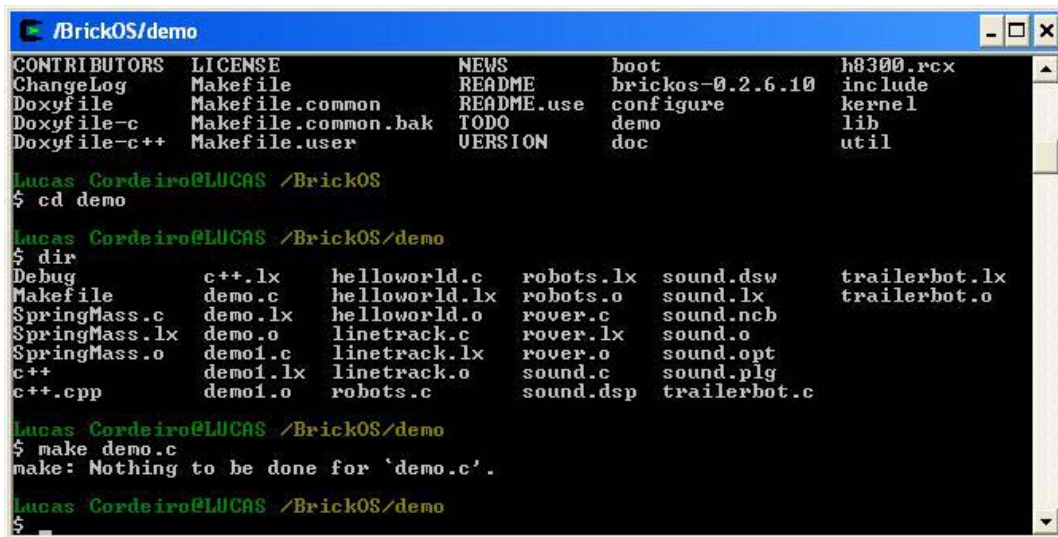
F.3 I cannot store other firmware

Whether the BrickOS has already stored in the RCX memory you need to remove the batteries to delete the firmware from memory. Other firmware cannot be stored in the memory if the previous one is the BrickOS.

The LeJOS only work if the RIS firmware that comes with the Lego Mindstorms kit is stored before. All firmware can be deleted from memory only removing the batteries.

F.4 My compiler does not work

In order to compile the BrickOS programs you need to follow the steps described in section 4.2.3. Whether you type the follow command `./dll ../Your Directory/Name of the file.c`, an error will appear because the file extension to be generated must be with the extension `.lx`. Error figure 5 depicts such situation.



```
Lucas Cordeiro@LUCAS /BrickOS/demo
$ cd demo
Lucas Cordeiro@LUCAS /BrickOS/demo
$ dir
Debug          c++.lx      helloworld.c  robots.lx    sound.dsw    trailerbot.lx
Makefile       demo.c      helloworld.lx robots.o     sound.lx     trailerbot.o
SpringMass.c  demo.lx     helloworld.o  rover.c     sound.ncb
SpringMass.lx demo.o      linetrack.c   rover.lx    sound.o
SpringMass.o  demo1.c    linetrack.lx  rover.o     sound.opt
c++            demo1.lx   linetrack.o   sound.c     sound.plg
c++.cpp       demo1.o    robots.c      sound.dsp   trailerbot.c

Lucas Cordeiro@LUCAS /BrickOS/demo
$ make demo.c
make: Nothing to be done for `demo.c'.

Lucas Cordeiro@LUCAS /BrickOS/demo
$
```

Error figure 5 – My compiler does not work

F.5 My robot does not work as desired

To solve this problem you must to follow the instructions of the subchapter 4.3.1.