THE DEVELOPMENT OF A VISUAL PROGRAMMING ENVIRONMENT FOR MOBILE ROBOTS

Dr. Antonio Heronaldo de Sousa
DEE - CCT - UDESC (State University of Santa Catarina), Post office box 631, Joinville – SC – Brazil
heron@joinville.udesc.br

PhD. Marcelo da Silva Hounsell
DCC - CCT - UDESC (State University of Santa Catarina), Post office box 631, Joinville – SC – Brazil
marcelo@joinville.udesc.br

Eng. Fabrício Noveletto
DSI - ACE (Education Association of Santa Catarina), Post office box 222, Joinville – SC – Brazil
fabricio@aceadm.com.br

Abstract. This paper presents the development of a programming environment for mobile robots, called K++. The object-oriented programming paradigm was used as the basis for development altogether with visual programming. The main characteristic of this environment is the fact that it allows the use of graphical and textual structures to best represent data and algorithms. K++ combines features of the object-oriented programming paradigm such as reusability, and features of the visual programming paradigm, such as readability. The use of object-oriented visual structures improves the quality and the availability of the information in the development of complex high-level algorithms for mobile robots applications. Also, the K++ environment allows on line simulations by using a specific "class" that was developed to implement the communication with the mobile robot. Preliminary tests using K++ in developing complex algorithms were greatly satisfactory and promising.

Keywords. *Object-oriented programming, visual programming, mobile robot, simulation, Khepera robot.*

1. Introduction

Mobile robots are ever more present at automated companies making it necessary to have new tools capable of extracting the best of these machines. Given the increasing number of applications and studies involving mobile robots programming, improvement on existing tools and the development of new ones are necessary. This paper will present a visual programming environment called K++.

The Khepera mobile robot was used for this work, which is a robot widely used in the academic and scientific community. The great advantage of using tiny little mobile robots (such as Khepera) for the study and the development of algorithms comes exactly from its size and reduced cost, allowing the creation of more easily and inexpensive simulation environments (Mondana et all, 1993).

Usually, the programming of this type of robot is accomplished in a textual way through the GNU C compiler or with the use of visual programming environments such as LABVIEW®, MathLab® (K-Team, 1999b) and Webots (Cyberbotics, 2003).

Using basic textual programming requires going into low level details and intricacies as well as a great deal of involvement and knowledge of the C language. However, the use of the visual programming environments makes it difficult to achieve full flexibility, as they are closed-source code software, once one can only use the libraries available in those environments, which is a very limiting aspect for the application development. In this sense, K++ as an open-source software allows future modifications to be made according to project's needs. K++ was developed using Microsoft Visual C++ (Kruglinski, 1996) as the programming language, which is an object-oriented visual programming environment. The existing visual programming environment in Visual C++ let the modifications in K++ to be made in a more productive way. Opposite to Visual C++ that is a general-purpose environment, K++ is an environment targeted at the development of mobile robot applications. In K++, visual and object-oriented paradigms have also been merged (as in Visual C++), making it possible to re-use of functions and to minimize the programming mistakes, guaranteeing a larger degree of productivity in the development of robot applications.

The developments of mobile robots applications using K++ make it possible to hide low-level details and therefore, focus the programming task at the application level such as algorithms for trajectory control. The prototype under development already allows the use of LOGO-like basic movement methods and other more complex methods for environments exploration.

2. The Khepera Robot

Khepera is a tiny little mobile robot that possesses similar functionalities to much larger mobile robots. The robot's basic module measures only 55mm of diameter and 30mm of height. Khepera is a sophisticated robot that incorporates a 32 bits processing unit working at 16 MHz of frequency and possesses 256 Kbytes of RAM memory as well as an EPROM memory that can vary from 128 to 256 Kbytes in size, depending on the robot's model. This is the basic module configuration that can be upgraded to include vision modules, radio controls and handlers/manipulators. A fully configured version of Khepera, including all above mentioned modules but radio control, running in autonomous mode can be seen in Fig. (1). The Khepera robot has interesting features for communication, motors control and sensoring.

Figure 1. Khepera robot.

2.1. Communication

Khepera can be commanded through a RS-232 serial PC connection or run autonomously. Usually, Khepera is kept connected to a PC at the implementation and program test phases. Afterwards, the application can be transferred directly into the robot's memory, making it possible to be run autonomously, in other words, without being connected to PC.

The communication between the host computer and the Khepera robot is made sending and receiving ASCII messages, could transfer data up to 38 Kbps. Every interaction is composed by: a command, sent by the host computer to the Khepera robot and followed by a carriage return or a line feed. When needed, a response, sent by the Khepera to the host computer. In all communications the host computer plays the role of master and the Khepera the role of slave. All communications are initiated by the master and is based on two types of interactions: one type of interaction for the set-up of the robot (for instance to set the communications parameters), and one type of interaction for the control of the functionality of the robot (for instance to set the speed of the motors and to get the values of the sensors). (K-Team, 1999b; and K-Team, 2002a)

2.2. Motors Control

The robot's movement is obtained through two DC servomotors coupled to reduction gearboxes (25:1) with a sophisticated sensor system. Incremental sensors are realizing with magnetic sensors and provide quadrature signals with a resolution of 600 impulsions per wheel revolution (Mondana et all, 1993). Furthermore, Khepera has a rechargeable battery system that allows 30 minutes of continuous movement in the autonomous mode. Motor control is achieved through the speed and positioning control. The motors speed is a value set in a pulse per 10 milliseconds, which corresponds to 8mm per second, reaching a maximum speed of 127 pulses per 10 milliseconds which is exactly 1 meter per second.

The Khepera main processor directly controls the motor power supply and can read the pulses of the incremental encoder. An interrupt routine detects every pulse of the incremental encoder and updates a wheel position counter. The motor power supply can be adjusted by the main processor by switching it ON and OFF at a given frequency and during a given time. The basic switching frequency is constant and sufficiently high not to let the motor react to the single switching. By this way, the motor react to the time average of the power supply, which can be modified by changing the period the motor is switched ON. This means that only the ratio between ON and OFF periods is modified. This power control method is called "pulse width modulation" (PWM). The PWM value is defined as the time the motor is switched ON.

Both motors can be controlled by a PID controller execute as an interrupt routine of the main processor. Every term of this controller (Proportional, Integral, Derivative) is associated to a constant, setting the weight of the corresponding term: Kp for the proportional, Ki for the integral, and Kd for the derivative.

The motor controller can be used in two control modes: the speed and the position modes. The active control mode is set according to the kind of command received. Different control parameters (Kp , Ki and Kd) can be set for each of the two control modes.

If the controller receives a speed control command, it switches to the speed mode. If the controller receives a position control command, the control mode is automatically switched to the position mode. Used in speed mode, the controller has as input a speed value of the wheels, and controls the motor to keep this wheel speed. The speed modification is made as quick as possible, in an abrupt way. No limitation in acceleration is considered in this mode.

Used in position mode, the controller has as input a target position of the wheel, an acceleration and a maximal speed. Using these values, the controller accelerates the wheel until the maximal speed is reached, and decelerates in order to reach the target position. This movement follows a trapezoidal speed profile. The input values and the control mode of this controller can be changed at every moment. The controller will update and execute the new profile in the position mode, or control the wheel speed following the new value in the speed mode.

The status of the controller indicates the active control mode, the phase of the speed profile (on target or in movement) and the position error of the controller.

Also, the PWM configuration parameters that control the motors can be easily altered, modifying their acceleration and deceleration. Fig. (2) illustrates the motor controls as well as the configuration access levels. It is possible to identify dark arrows in Fig. (2) coming from the left-hand side that are the "user control modes" which are divided in "PWM control" (acceleration and deceleration control), "speed control" (velocity control) and "target position control" (for distance controlling).

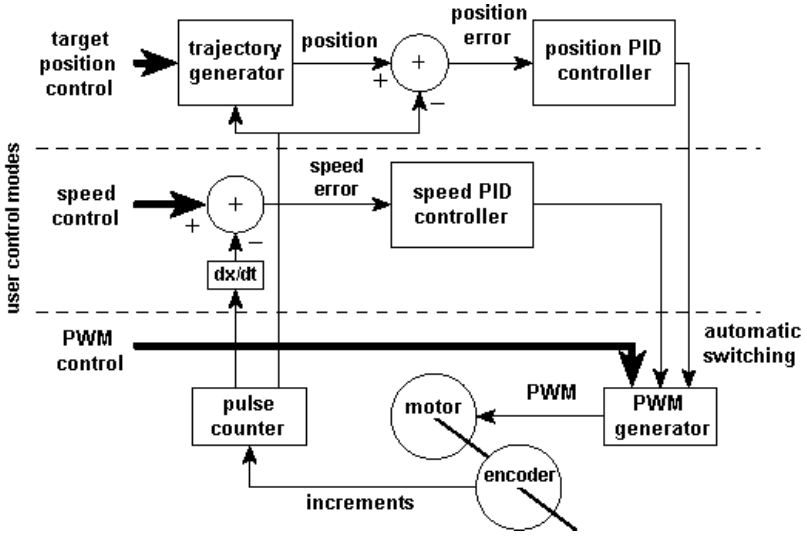

Figure 2. Robot motors controls and the access configuration levels.

For the robots positioning control, the feedback unit is equivalent to a pulse at the encoder, which corresponds to 0,08 mm. Therefore, a request to the robot to "walk" a specific distance must be converted into pulses. (K-Team, 1999b).

2.3. Sensoring System

The sensoring system is composed of eight infrared sensors (IR) with source and receiver in the same component. Every IR sensors works in obtaining two different measurements: objects proximity and ambient illumination (or brightness) levels. The Infrared sensors are disposed in appropriate angles as shown in the fig. (3), in a way to cover the field of the robot's performance in a satisfactory way.

The sensibility of the proximity sensors is directly related to the type of reflective material. The proximity sensibility scale possesses a resolution of 10 bits (values between 0 and 1023), the higher the value the closer the robot is to an obstacle. These values also depend on the reflectivity of the obstacle. The best the reflectivity the higher will be the sensibility of the sensor. A white plastic, for instance, has the greatest reflectivity, while a dark wood has the lowest reflectivity.

The brightness sensors work in a similar way to the proximity sensor with the same sensibility scale, being 450 the average value for a dark environment. In this way, the smaller the value returned by the ambient light sensor, the closer top the light source the robot is (K-Team, 1999b and K-Team, 2002a).

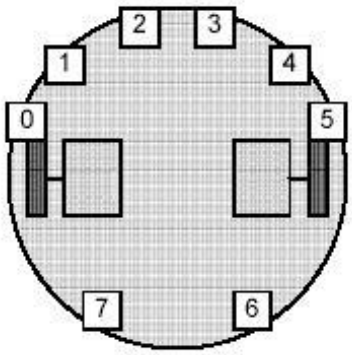

Figure 3. Infrared sensors positioning.

With all those functionalities, the Khepera robot seems to be the most suitable simulation environment to test complex algorithms developed for trajectory planning, obstacles avoidance, sensorial data processing, among many other applications (Floreano et all, 1996).

However, controlling the robot at the low-level shown needs a great deal of attention from the application programmer for it uses pulses as basic unit (even for positioning) requiring continuous unit conversion and parameter with not very straightforward meaning (such as Ki, Kd and Kp). Furthermore, in most cases low-level details such as these do not help solve real life tasks and thus, a higher level of programming is required.

At this higher level of programming, intricate low-level parameters can be set at default values (for instance, Ki, Kd and Kp are left at their default values for most applications) but altered if necessary. This would facilitate a lot the task of solving a real-life application.

This paper is focused in showing the importance, productivity and capabilities of a high level programming approach and thus, access to low-level parameter configuration although easily conceived and implemented, will be left out of the discussion.

3. The Conventional Approach

The Khepera robot can be programmed in several ways. The most common one is to send to the robot commands in a one-by-one basis through a specific communication protocol, using any serial communication software. It seems clear that this approach make it impossible to implement any algorithmic solution to a “pragmatic” application because they cannot manipulate the data sent to nor received from the robot. So, in order to allow the robot to perform a (higher level) task the robot needed a more sophisticated type of programming. Among known conventional programming environments the following can be included:

Khepera KTProject, Windows™ graphics interface developed by the robot's manufacturer (K-Team, 1999b). KTProject allows the development and compilation of a C program to be downloaded afterwards into Khepera´s memory, allowing it to execute tasks in an autonomous mode.

Another important programming tool is the GNU C Cross Compiler that allows the development of native applications in Unix/Linux platforms.

There are also commercial software like LabVIEW® (National, 2000) and MATLAB® (K-Team, 1999b) that have specific libraries targeted for Khepera which allow the development of more complex applications. However, the area of the robot's performance is limited once it stays connected to the PC through a serial communication cable that links the robot to PC (tethered mode).

Another example of a programming environment is the Webots (a commercial simulator by Cyberbotics), which allows the user to see a virtual 3D robot moving about as well uploading the simulation results to the real Khepera robot (Webots, 2003). It is important to point out that Webots uses the visual paradigm only in terms of a graphical 3D virtual world where a virtual robot can be found. However, the algorithms that control the robot (either the virtual or the real one) are described using conventional textual C language.

On facing these serious limitations K++ was implemented as a research test bed, developed from scratch but open source. All the intrinsic knowledge gathered from developing K++ is of vital importance once it is an academic software that is being fine-tuned continuously in order to extend K++ programming environment to other applications and robots.

4. The Object-Oriented Programming Paradigm

Even after the appearance of new programming tools, a good programmer should have intelligence and skills to find and to use abstractions, experience and creativity. In this sense the object-oriented programming paradigm is more than simply a collection of new resources added to an existing programming language. The object-oriented programming is an advanced programming paradigm that represents a new way of organizing knowledge (Lee et all, 2002; Voss, 1991).

The main concepts of the object-oriented programming paradigm are:

Object: An object is a software abstraction that can represent something real or virtual. An object is composed of "members of data" and "methods" that manipulate these "members of data". Therefore, objects can be characterized by their private attributes (members of data) and by their behaviors (methods). Methods are run in order to accomplish some task such as altering the contents of an object's member of data, changing its status and/or behavior.

Class: A class is the main concept of the object-oriented programming paradigm and consists of "members of data" and "methods" that represent features of a group of similar objects. Therefore, a class represents a template whereby objects can be created (or instantiated). In other way, when a group of objects presents the same characteristic, there will exist at least one class where those objects must belong to. It is necessary to emphasize that the classes are static entities and can only be modified at compilation time while objects are dynamic entities that exist in memory and which only have their "members of data" altered and their "methods" executing a task during execution time. A class effectively allows the re-use of code.

Messages: Objects only communicate through "message passing". It can be said that a message is a requests for an object to alter its status or execute a task. In other words, it can be said that a method of the object that received the message is executed based on the incoming parameter or based on the object's own members of data.

Encapsulation: The members of data of a specific object can only be altered by pre-defined methods of its own class. The only way an object can alter the "members of data" or "behavior" of another object is through sending a message to the other object to activate a method that is able to do this. The concept where attributes and methods are only visible through message passing is called as encapsulation. Encapsulation works as a protection to the members of data and methods beyond the fact that it turns explicit any kind of communication with the object.

Hierarchy: The hierarchy concept allows the definition of a new class based on an already existing one. Inheriting members of data and methods can facilitate the creation of sub-classes. Therefore, new classes can be created by specialization of the behavior of (master), leading to a hierarchy of classes.

Polymorphism: The term polymorphism is used in biology to identify shape and functions variations of the same specie. In object-oriented programming polymorphism means almost the same, i. e., it is a way to allow objects variation of the same class to respond to an unified message but to function differently according to some specialization (if there is one).

The use of the object-oriented programming paradigm allows productivity increase due to a better structure of the application (thanks to classes and objects) and to the re-use of software elements (thanks to hierarchy and encapsulation); all of this in a very consistent way (thanks to polymorphism and message passing). It also complies the programmers' psychological characteristics, because as far as learning theories are concerned it is said that the organizing incoming information helps achieve a steeper learning curve and a better memorization (Sousa, 1999). In addition, the information recovery becomes much easier if information is organized from the beginning.

Furthermore, items in long-term memory are much more quickly recovered the more they are structured and classified. The strategies of existing methods for memory development consist in learning how to organize things that should be learnt so they can be easily found in our memory, when necessary.

These assertions go with the structuring benefits that object-oriented programming can promote: The objects (entities) are organized or classified into classes, building up a well-organized relationship structure.

5. The K++ Programming Environment

The K++ environment was designed based on object-oriented paradigm. It uses a subset of textual declarations in C++ (which is an object-oriented language), introduces graphic symbols to represent classes and has support to the structured programming based on Nassi-Shneiderman graphic arrangement technique, known as NS charts (Nassi et all, 1995). The use of the NS charts allows the construction of well-structured programs, substituting the unconditional deviations for nesting arrangements. This chart uses visual structures to represent the flow control and textual information to describe the manipulated data, tests of conditions, and indication of number of repetitions, among others.

It must be clarified that the development of K++ made use of Visual C++ aid to build up the graphic user interface (buttons, menus, slides, combo boxes, etc.) but the resulting application is a completely different sort of "visual" environment. One way of using Visual C++ is by select pre-defined visual elements, which the user places in the screen by use of a "wizard-like" resource and "fills the blanks" adding functionality to the visual elements. In K++ the visual elements represent a new way of expressing and documenting programs and the user can start it from scratch if necessary but the user can also make use of pre-defined 'classes" of visual entities that have close relationship to mobile robots.

The K++ programming environment was developed with the objective of minimizing the limitations in current environments. The central idea of K++ is to allow it to develop applications in a more efficient way. This is different to the

development using native applications that requires highly skilled programmers. This also minimizes knowledge requirement of LABVIEW® and MATHLAB® software and their limitations, which are related to the use of libraries and pre-defined functions. The main characteristic of the K++ environment is the diversity of functions and applications that can be developed for the robot beginning from classes that implement various methods for controlling the robot.

Figure (4) illustrates K++ environment. It was designed to work with three frames.

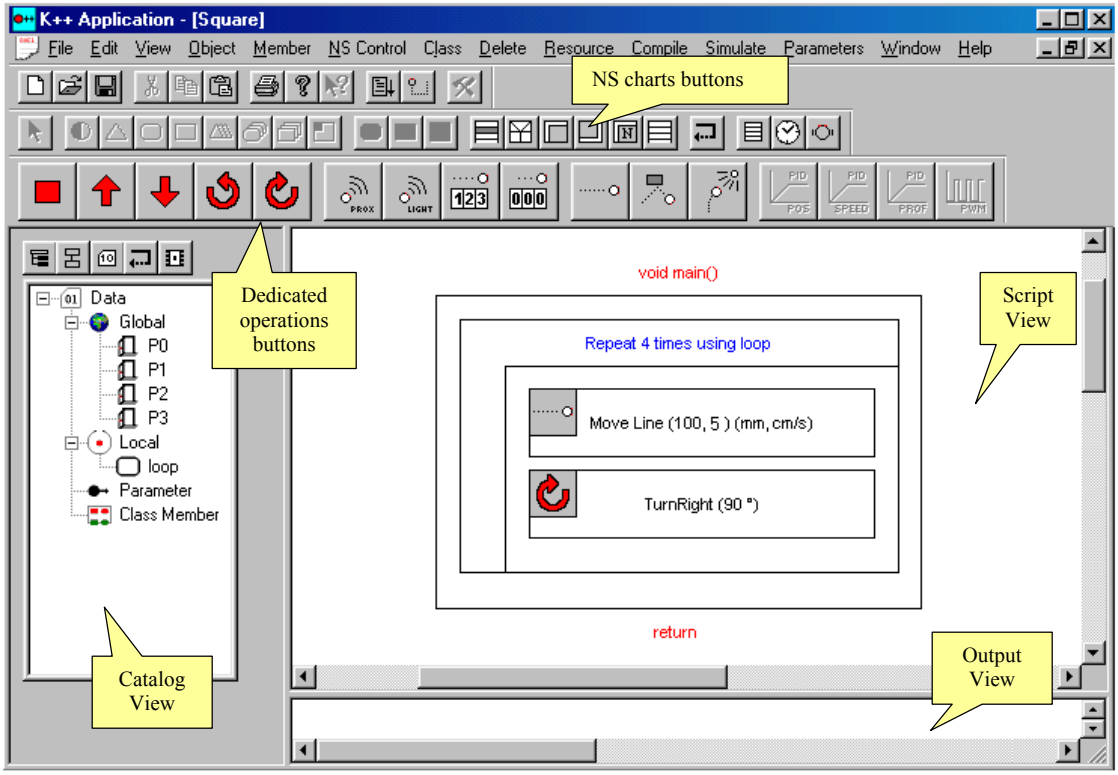

Figure 4. K++ programming environment

The first, called “catalog view” (left side), is used to organize the program elements (classes, code references and data members). The second frame, called “script view” (right hand), is used for code drawing, classes and data descriptions. The last, called “output view” (at the bottom of the right hand side), is used to show error messages and warnings.

K++ offers a toolbar that facilitates choosing operations implemented using Khepera class that the user wants to utilize.

To promote ease of use and to reduce the users mistakes, K++ man-machine interface uses two important mechanisms: the automatic visualization of the elements at the “catalog view” and the easy information exchange among the environment frames (by using drag-and-dropping).

K++ environment does not demand the applications to be implemented using the object-oriented programming paradigm. It is important to point out that, despite of the fact that K++ was implemented using the object oriented paradigm and the fact that it permits the use of object-oriented concepts, it is also possible to use K++ to deploy a purely structured procedural implementation (i.e., an object-based implementation). This is so because K++ environment is also targeted to application programmers that have actually no knowledge about object-oriented programming. Consequently, the range of programmers that can use K++ language is widened (although all K++ benefits are not reached).

6. The Khepera Class

K++ environment offers a group of dedicated operations that allows the development of a wide variety of applications for the Khepera robot. The use of these dedicated operations group facilitates the implementation of the applications, minimizes programming mistakes and accelerates the development of the applications. At the moment, available operations include the main functions to control the robot, such as to walk, stop, rotate and read proximity and brightness sensors. Those operations are obtained by dragging it from the toolbox into a proper position on the script view, which means that a message will be passed to the Khepera class in order to invoke that specific method.

The Khepera class was developed to allow K++ environment programming to simulate algorithms once it is compiled and it also generates code for the robot to work in autonomous mode. To this aim, it uses a communication protocol developed by the K-Team specifically for Khepera robot where commands are sent to the robot and received data are processed. Thus, the Khepera class implements the serial communication control (Campbell, 1993) and it implements several methods to manipulate the robot.

Implemented methods are divided into two different levels of abstraction and are described in Tab. (1) and Tab. (2). The robots basic methods, seen in Tab. (1), were named according to the LOGO language (Harvey, 1997). LOGO was used as a reference because there is a close relationship the way a mobile robot such as Khepera and a "virtual turtle" moves about. Furthermore, it is believed that all teaching-learning background and benefits behind LOGO are somehow incorporated into K++ by using the same set of operations (Papert, 1985).

Table 1. Basic methods for the robots control

Basic Methods	***Methods description***
MoveForward ( )	Moves robot forward with specific speed in cm/s
MoveBack ( )	Moves robot backwards with specific speed in cm/s
TurnLeft (degrees)	Rotates robot to the left on its axis by the specific degrees
TurnRight (degrees)	Rotates robot to the right on its axis by the specific degrees

Table 2. Complementary methods for the robot control

Complementary Methods	***Methods description***
SetSpeed (speed)	Sets speed motors to both motors
SetPIDspeed (Kp, Ki, Kd)	Set PID speed controller
SetPIDposition (Kp, Ki, Kd)	Set PID position controller
ReadLightSensors( )	Reads brightness sensors
ReadProxSensors( )	Reads proximity sensors
ReadMotorPosition( )	Reads position of motors M1and M2
StopMotors( )	Stop both motors
ResetCountMotors( )	Reset count motors encoder
MoveLine (distance, speed)	Moves robot forward with specific distance and speed
LightFinder( )	Rotates robot in the light source direction
ObstacleSkipper (degrees)	Rotates and deviates robot when finds obstacle

Using the methods described in Tab. (1) and Tab. (2) it is possible to implement any kind of robot movement and read any of the robot sensors. Thus, it is possible to develop a wide variety of applications such as algorithms to follow paths (Bianchi et all, 2001) and algorithms for Breintemberg vehicles (Breintemberg, 1984), among others. Furthermore, because the Khepera class is implemented using the object-oriented paradigm, it allows its code to be re-used for some other Khepera application developed in Visual C++.

It is important to stress out that the programmer that uses the Khepera class does not need to know how each one of the methods is implemented, but is suffices to know how the user can re-use it in order to build new methods and to develop a great number of complex algorithms based on already existing methods.

7. Simulation and code generation in K++

The simulation within K++ is an important tool for algorithms testing. This feature allows error debugging "on the run" (alongside with the development process), before generating code and compiling it to download it to the robot. The final code generation should be made after the algorithm was simulated with success. The basic idea of simulation is to test the application but using the robot attached to the to the PC through serial communication cable so the robot can execute the whole application still being developed. Code generation, on the other hand, allows the (final and resulting) application to be downloaded into de robot and then, it would execute its task without the need of a serial communication and thus, in an autonomous mode. In the code generation phase, K++ environment translate the whole visual structure to a standard C file (Holzner, 2001). In this file, the methods used in the application are translated by K++ environment to existing commands in the BIOS of the Khepera robot (K-Team, 1999a; K-Team, 2002b).

The resulting C code is loaded to a specific Khepera robot compiler that translates the code to the S-Motorola file format (K-Team, 1999b) after which the final result can be uploaded in the robot. Once the application is loaded in the robot's memory, the robot is ready to run the application in the autonomous mode.

In the "script view" of fig. (4) there is an example of a very simple application developed in K++, where the robot makes a square trajectory. The NS charts in Fig. (4) represent an iteration command ("for" command) that will repeat the set of commands within it four times. Thus, the program starts up by activating the method *MoveLine*, which instructs the robot to walk in a straight line for 100 mm at 5 cm/s speed (default speed). Afterwards the method *TurnLeft* is activated, commanding the robot to rotate 90 degrees to the left on the robot's own axis. Repeating this same procedure 4 times will produce a square path. This example shows that the use of graphic arrangements in the development of applications is quite clear once it is believed that visual programming is much more readable than the textual programming.

The main feature of K++ programming environment is that it allows the use of objects (from the object oriented infra-structure) within algorithms that are described visually using NS charts (therefore, this is visual paradigm implied here), which describe. The use of oriented object programming is suggest for the development of complex algorithms, where the reuse of code essential. Figure (5) shows the main modules of an application where one can see two objects, called AGV and AGV_LIGHT, belonging the *Explorer* class (see Fig. (6)) and *LightFinder* class (see Fig. (7)), respectively. The *Explorer* class aims the environment and the *LightFinder* class, a specialization of the *Explorer* class, has the objective of navigating (to explore) in the direction of the brightest light source.

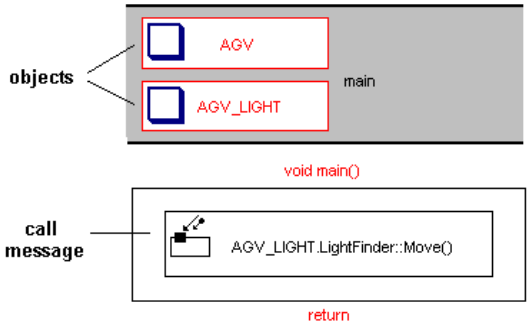

Figure 5. Main module description.

The *Explorer* class (see Fig. (6), top right-hand side) has two methods: the *Move* and *Navigation* methods. The *Move* method (bottom left-hand side of Fig. (6)) it executes two operations: firstly it executes the *MoveForward* operation to make the robot to move straight ahead, afterwards it executes the *ObstacleSkipper* operation, which identifies and deviate from obstacles.

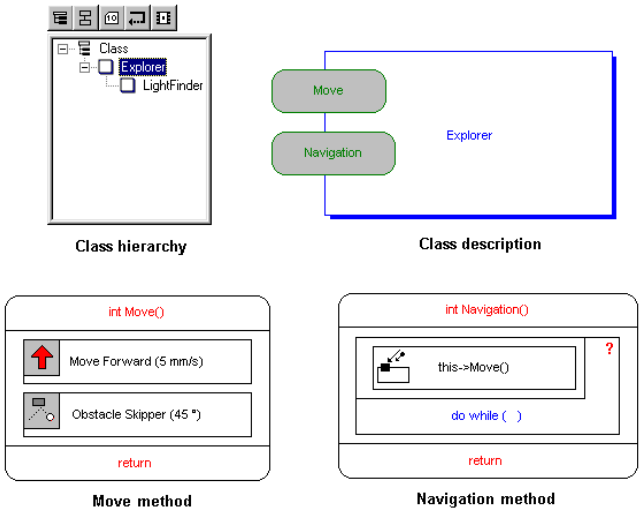

Figure 6. Explorer class.

On the other hand, the *Navigation* method (bottom right-hand side of Fig. (6)), executes an eternal loop due to a control structure of the do/while type, where a call for the *Move* method (of the father class, *Explorer*) is done.

The *LightFinder* class showed in Fig. (7) possesses only the *FindLight* method. This method executes (see right-hand side of Fig. (7)), within a control structure of the do/while type, the operation *LightFinder*, which verifies the light intensity, in each sensor, and rotates the robot in the direction of the brightest light source, returning the smallest value registered by the sensors. It is important to point out that the brightest light source the smaller value registered by the sensor. In the *FindLight* method, the do/while loop is halted when the brightness given by a sensor is less than the saturation value.

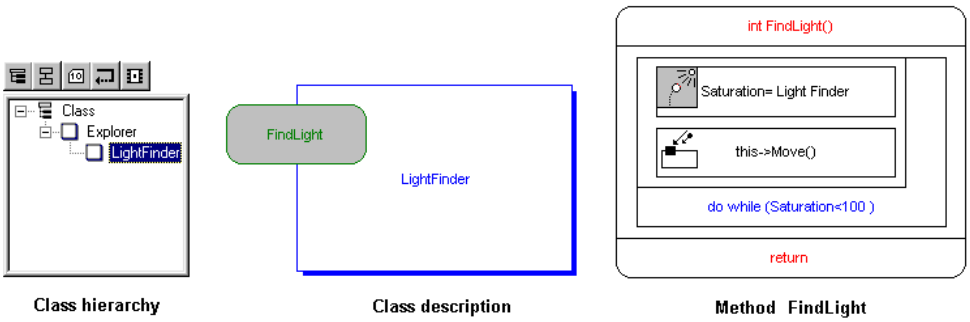

Figure 7. *LightFinder* class.

This example highlighted the object oriented programming advantages. The creation of a new class (*LightFinder* class) starting from an already existing class (*Explorer* class) is done by code reuse thanks to the inheritance of the methods available in the father-class. The objects manipulation is done by sending messages, which facilitates the understanding of the developed algorithms.

8. Conclusion

The present work is an answer to a concrete need in the development of specific software in the area of mobile robotics. Up to now, a lot of time is still lost in the study and development of applications for mobile robots. A lot of trial-and-error approach is still a common practice. It is not uncommon to see programmers developing a whole application, uploading it in the mobile robot just to see what happens. With the use of a programming environment such as K++ the gap between projects phases can be sensibly minimized. K++ environment aims to high productivity, much thanks both to the object-oriented programming paradigm and the flexibility of the visual programming.

The proposed environment has the benefit to be used by non-specialist C programmers. K++ environment also has a friendly graphical user interface that facilitates the programming, development and simulation of many applications. The choice of the object-oriented paradigm and the Visual C++ programming language for implementing K++ environment has been proved to be very efficient.

To conclude, it should be stressed out that other important feature of the K++ is the possibility to develop new tools for other mobile robots that run C programming language at the core level. This is possible simply by designing a new subclass that complies with the new robot's functional aspects. In this sense, the K++ visual programming environment can extend its benefits to a large group of different mobile robots.

9. References

Bianchi, R.A.C., Simões, A.S., Costa, A.H.R., 2001, "Reactive Behaviors to Follow Tracks in Guided Mobile Robot to put Vision", (in Portuguese) V Brazilian Symposium of Intelligent Automation. 6 p. Also available at http://www.lti.pcs.usp.br/~rbianchi/publications/sbai2001-usp.pdf

Braintemberg, V., 1984, "Vehicles: Experiments in synthetic psychology", MIT Press, Cambridge, pp. 1-32.

Campbell, J., 1993, "C Programmer's Guide Serial to Communication", Sams, pp. 407-598.

Cyberbotics, 2003, "Webots Reference Manual", Cyberbotics, 64 p. Also available at http://cyberboticspc1.epfl.ch/cdrom/common/doc/webots/reference/reference.pdf

Floreano, D. and Mondana, F., 1996, "Evolution of homing navigation in a mobile real robot", IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26, pp. 396-407.

Harvey, B., 1997, "Computer Science Logo Style", MIT Press, Vol. 1, Cambridge, pp. 17-39.

Holzner, S., 2001, "C++ Black Book", Makron Books, 646 p.

Kruglinski, D.J., 1996, “Inside Visual C++”. Microsoft Press.
K-Team, 1999a, “Khepera BIOS Manual”, K-Team, Switzerland, 121 p. Also available at http://www.k- team.com/download/khepera.html
K-Team, 1999b, “Khepera User Manual”, K-Team, Switzerland, 52 p. Also available at http://www.k- team.com/download/khepera.html
K-Team, 2002a, “IR Sensors Report”, K-Team, Switzerland, 19 p. Also available at http://www.k-team.com/download/khepera.html
K-Team, 2002b, “Khepera Programming Manual”, K-Team, Switzerland, 48 p. Also available at http://www.k- team.com/download/khepera.html
Lee, R.C. and Tepfenhart, W.M., 2002, “UML and C++ - Practical Guide of object-oriented development”, Makron Books, pp. 23-39.
Lund, H.H., Cuenta, E.V. and Hallan, J., 1996, “A Simple Mobile Real-team Robot Tracking System”, Technical Paper 41, Department Artificial of Intelligence, University of Edinburgh.
Mondana, F. and Wrinkled, E. and Ienne, P., 1993, “Mobile Robot Miniaturization: A Tool For Investigation In Control Algorithms”, Experimental Robotics III - Proceedings of the Third International Symposium on Experimental Robotics, October 28-30, 1993. Kyoto,Japan, pp. 501-513.
Nassi, I. and Shnneiderman, B., 1984, “Flowchart Techniques for Structured Programming”, ACM Sigplan Notices, Vol. 8, no. 8, pp. 723-728.
National, 2000, “LABVIEW® Manuals”, National Instruments, 272 p.
Papert, S., 1985, “LOGO: Computers and Education”, (in Portuguese) Ed. Brasiliense, 254 p.
Sousa, A.H., 1999, “A proposal of object-oriented visual language for microcontroller programming”, Doctorate Thesis (in Portuguese) – Faculty of Electric Engineering - State University of Campinas (UNICAMP), 116 p.
Voss, Greg., 1991, “Object-Oriented Programming: An Introduction”, McGraw-Hill, 584 p.