Reflections on Using Robots and Visual Programming Environments for Project-Based Teaching

I. Plauska[1], R. Lukas[1], R. Damasevicius[2]
[1]*Centre of Real Time Computer Systems, Kaunas University of Technology, Studentu St. 50, LT-51368 Kaunas, Lithuania*
[2]*Department of Software Engineering, Kaunas University of Technology, Studentu St. 50, LT-51368 Kaunas, Lithuania*
robertas.damasevicius@ktu.lt

***Abstract*—Visual programming languages provide a more natural approach to specifying software/hardware systems with complex behaviour such as robots. They are especially important in education because they do not require formal knowledge of programming language syntax and are attractive to users. We present an analysis and comparison of two visual programming environments, Lego NXT-G and Microsoft Visual Programming Language, based on the cognitive and usability requirements, evaluate their application in robotics-based Computer Science education, identify main problems and propose solutions for using visual programming languages in the Internet-of-Things domain.**

***Index Terms*—Electronic learning, graphical models, programming environments, robot programming.**

I. INTRODUCTION

Robotics is an exciting multi-disciplinary area that is rapidly advancing into our everyday life. Current university students will live in a digital society surrounded by industrial and service robots at work, educational robots at schools, medical and assistive care robots at hospitals, and entertainment robots at home. The educational priorities must shift towards teaching students how to manipulate all digital devices (smart phones, tablet PCs, smart TVs, robots, etc.) that surround them for their own needs. Robots can be seen as multi-faceted tools with different roles in engineering education as follows:

– *Robot as specialized computer* with both computational and mechanical facilities to perform physical movement-oriented tasks [1];
– *Robot as a mobile smart thing:* a mobile platform for wireless sensors in the context of the Internet of Things [2];
– *Robot as Learning Object* [3]: physical and reusable unit of knowledge that can be used to teach the concepts of programming, robot control and artificial intelligence;

This paper presents an analysis of educational visual programming languages in the context of robotics-based teaching, identifies main problems and proposes solutions for improving visual languages and extending them to the Internet-of-Things domain.

II. ANALYSIS OF VISUAL PROGRAMMING ENVIRONMENTS

Visual programming is a method to specify a program in a two (or more) dimensional fashion [4], whereas in a textual language a programmer writes and a compiler or interpreter processes a program as a one-dimensional stream of symbols. Visual programming language uses meaningful graphic representation and manipulates visual information or supports visual interaction [5] in the process of programming. Visual elements based on imagery thinking provide a more natural approach for specifying a program than textual languages and the 2D representation is more suitable for representing parallel behaviour of complex systems consisting of multiple components such as robots. Visual languages are especially important in education, because their do not require formal knowledge of programming language syntax, are visually appealing and attractive to their users, and can be combined with additional engagement-enhancing concepts such as *gamification* [6].

The Cognitive Dimensions introduced by Green and Petre [7] provide a framework for assessment of a programming system as follows: Closeness of mapping (closeness of programming structures to problem domain), Consistency (similar semantics are expressed in similar forms to allow inference), Error-proneness (possibility of making mistakes because of poor notational design), Hard mental operations (thought processes required to formulate an expression made difficult by the notation), Hidden Dependencies (important relationships between entities are not visible), Progressive evaluation (ability to execute the program partially, before all of it is put together), Role-expressiveness (purpose and role of each component is easily inferred), Secondary

Manuscript received March 14, 2013; accepted July 2, 2013.

The work described in this paper has been carried out within the framework the Operational Programme for the Development of Human Resources 2007-2013 of Lithuania „Strengthening of capacities of researchers and scientists" project VP1-3.1-ŠMM-08-K-01-018 „Research and development of Internet technologies and their infrastructure for smart environments of things and services" (2012-2015), funded by the European Social Fund (ESF).

notation (extra information other than program syntax, that conveys extra meaning beyond semantics), Visibility (ability to view parts of a program simultaneously and easily).

Other comparison criteria have been formulated by Howard [8]: Intended audience, Paradigm, Ease of use, Visual representation, Reusability, Data structures and types, Effective use of screen area, Effective use of colours, Clarity of graphical symbols, Interactive capabilities.

Next we analyse a subset of visual languages used in educational setting for teaching robot programming.

A. *Microsoft Visual Programming Language*

Microsoft Visual Programming Language (MVPL) is a part of Microsoft Robotics Developer Studio (MRDS). MVPL is a robotics-oriented data flow language that generates code to execute robotics programs designed with MRDS. In MRDS, every action is performed by a service, a basic building blocks that controls a particular software or hardware entity. Each service expresses an isolated behaviour that is defined independently and can interact with other services. MVPL provides 10 general-purpose building blocks, called *basic activities* (Fig. 1(a)) and many various application (or robot) specific blocks as well as blocks used to simulate hardware, called *services* (Fig. 1(b)). Some blocks have additional parameters that can be configured.

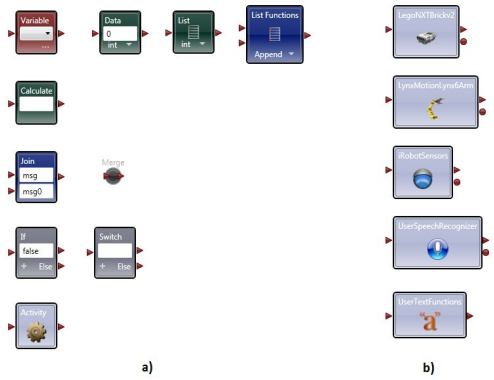

Fig. 1. Common VPL blocks.

The example program in Fig. 2 is written for the LEGO NXT robot to use ultrasonic sensor for collision avoidance while driving. The `NxtUltrasonicSensor` block sends the sonar data every time it gets a reading. The `If` block then activates one of three execution branches. The first branch is activated when the robot is really close to an obstacle; then, the `Data` block sends a defined rotation value to the `NxtDrive` block, which controls the robots driving motors. The robot is instructed to turn left 45 degrees. The second branch is activated when the obstacle is in mid-range. Two defined drive power values (0.3 left and 0.6 for right wheel) are joined to a named list by the `Join` block and sent to the `NxtDrive` block, forcing the robot to turn moderately left. The last branch is active when there is no obstacle detected, and the same drive power is applied to both wheels, forcing the robot to go ahead. Different types of control data can be sent to the same `NxtDrive` block. This is achieved by configuring the data connection. The `NxtBrick` block is not connected, but it is necessary in order to configure the NXT robot.

MVPL is simple enough even for a novice user to get started quickly, however, it has enough functionality to construct complex programs. Large number of additional services makes it possible to control many types of robots. Modelling capabilities make it easy to experiment even without having a real robot. However, the dataflow concept can make it harder to move to the sequential languages (like C++ or Java), which are *de facto* standard for programming.

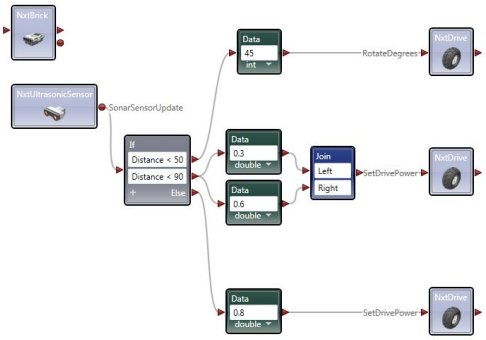

Fig. 2. Example of VPL program.

B. *Lego Mindstorms NXT*

NXT-G is a visual programming language and environment developed by Lego for Lego Mindstorms NXT robotics kit. The language is aimed at children as well as adults with no programming background. The language is block-based: each physical Lego block contains hidden digital implementation, and can be chained together to perform a sequence of behaviours. Sensor blocks receive inputs from outside world and send data to other blocks [9].

NXT-G blocks (Fig. 3) are of six categories: a) *Common* - action `Move` and other commonly used blocks from other groups; b) *Action* - general robot control actions such as `Motor control`, `Sound` or `Display`; c) *Sensor* - blocks that allow to control and get data from the sensors; d) *Flow* - flow control blocks such as `Wait` or `Loop`; e) *Data* - blocks that allow to define variables and perform mathematical or logical operations with them; f) *Advanced* - additional action blocks like `File access`. Each block can be additionally configured using a simple Configuration menu (see Fig. 4).

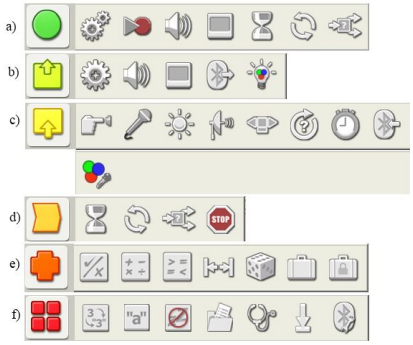

Fig. 3. Lego blocks and sensor blocks.

Fig. 4. Configuration menu of Move block.

Figure 5 shows a program which implements the same robot control algorithm using a supersonic sensor as in

Fig. 2 The main difference from MVPL is that all blocks in the diagram are invoked sequentially and there are flow control blocks such as `Loop` (similar to looping statement in textual languages). The program in Fig. 5 has one infinite `Loop` block which contains the remaining functionality. Inside the `Loop` block, there is a `Switch` block, which can be tied to a selected sensor (supersonic sensor in this example), reads and compares the data from that sensor to a defined value, selects one of two branches according to the result. As the `Switch` block only allows one comparison, two nested blocks are required in order to define three branches similar to Fig. 2. The `Move` blocks control the drive powers of two motors for steering the two-wheeled robot.

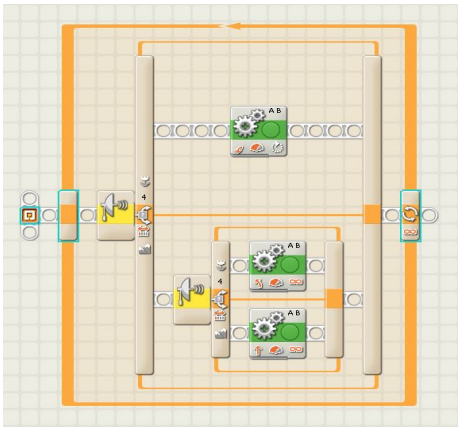

Fig. 5. Example of NXT-G program.

As NXT-G is aimed specifically at programming the Lego NXT robot, most of the blocks have a clearly defined functionality which directly translates to concrete actions of the robot which can be visually observed. This makes programming straight-forward as compared to more abstract MVPL, and more suited to novice users. The sequential execution and flow control of NXT-G is closer to traditional text programming languages than MVPL and could make it easier to move to mainstream programming.

The disadvantage of NXT-G is its overtly iconographic representation: NXT-G displays the numerical or logical values of the blocks by icons or hides them altogether. To see concrete values one has to click each block and open the configuration window. While this might be more visually appealing to intended users, it is harder to make sense of larger programs with many defined values. Other disadvantages are the representation of the FOR loop and poor usability of some block icons.

C. *Evaluation*

The evaluation of MVPL and NXT-G is summarized in Table I. Qualitative criteria are evaluated using the 4-level measurement scale (none, weak, fair, strong).

III. VISUAL PROGRAMMING IN ROBOTICS PROJECTS

The use of visual programming languages was explored during laboratory works of "Robot Programming Technologies", a course delivered at KTU Faculty of Informatics to 4th year bachelor students.

The course was attended by 34 students. Course aims to teach students of the basic principles of robot control and robot programming. The main concepts to learn are state (property of the robot), action/reaction (change of the state of the robot due to external or internal factors), behaviour (specific sequence of actions aimed to achieve a pre-set objective), decision (ability to undertake a specific sequence of actions from a set of alternatives), autonomy (ability to function independently), communication (ability to send/receive messages from external devices).

TABLE I. EVALUATION OF MVPL AND NXT-G LANGUAGES.

Criterion	Lego NXT	VPL
Intended audience	Children, novice robot programmers.	Novice programmers. Useful for experienced programmers in rapid prototyping.
Paradigm	Sequential statements.	Data flow.
Ease of use	**Fair.** Fairly easy, intended for novice programmers.	**Weak.** Requires understanding of parallel programming concepts.
Visual representation	Rectangular blocks, with their functionality and parameters indicated by icons.	Rectangular blocks, with their functionality and parameters indicated mostly by text.
Reusability	User can create reusable blocks with a program inside them.	Reusable activity blocks can have a program inside them.
Data structures and types	Basic variables (no arrays) and types (integer, Boolean).	Basic variables and arrays can be created, many data types.
Effective use of screen area	**Weak.** Blocks can only be stacked horizontally and there are no scrollbars, so larger programs spread beyond the screen bounds	**Fair.** Most of screen area can be used to build a program, blocks can be freely arranged and easily placed anywhere in the programming window
Effective use of colours	**Fair.** Colours are used to indicate blocks belonging to the same category.	**Fair.** Colours are used to indicate blocks of similar or close functionality.
Closeness of mapping	**Strong.** A robot specific language.	**Fair.** A multi-purpose language
Clarity of graphical symbols	**Weak.** Heavy use of icons makes it harder to immediately understand function of used blocks.	**Fair.** Symbols are simple and easy to understand. Basic functions of blocks are indicated by text.
Closeness of mapping	**Strong.** Most language blocks map directly to concrete capabilities of the NXT robot.	**Fair.** Many services map directly to concrete capabilities of different robots, however, basic activities are abstract.
Consistency	**Strong.** Blocks from the same group are used consistently	**Weak.** Similar services for different robots can be used very differently.
Error-proneness	**Strong.** The simplicity of the language doesn't allow for many slips.	**Fair.** Mostly textual data and parameters representation can cause typos.
Hard mental operations	**Fair.** Programing is mostly straight forward; however, more complex programs can be a challenge.	**Fair.** Although there are more abstract blocks, it can make programing complex operations easier.
Hidden Dependencies	**Strong.** Dependencies in lower levels are made obvious by icons in the highest level.	**Weak.** The functionality and inner dependencies of most of the services is not immediately obvious.
Progressive evaluation	**Fair.** Created programs can be encapsulated in special block and reused later.	**Fair.** Created programs can be encapsulated in special block and reused later.
Role-expressiveness	**Strong.** Roles of most of the blocks are obvious.	**Fair.** Basic activities are abstract. Many services express concrete roles.
Secondary notation	**Fair.** Variables can be named; much of extra information is represented by icons. Comments can be written in program window.	**Fair.** Expressed mostly in text; however, this can make complex programs more understandable than in NXT-G.

The robot hardware used during lab works included two LEGO NXT robots with NXT Intelligent Brick, a brick-shaped computer (32-bit ARM7, 256 KB flash memory, 64 KB RAM, and secondary 8-bit AVR with 4 KB flash memory, 512 B RAM) with 100 × 64 pixel monochrome LCD display and 4 buttons to navigate a user interface. The Brick can control up to three motors and input from four sensors (light, sound, ultrasonic, touch) via RJ12 cables. The Brick also has built-in Bluetooth for transfer of programs and data.

The Arduino 4WD Mobile Platform provides a 4 wheel drive system with ATmega328 microcontroller board and 4

DC Motors. The platform can be connected to a computer with a USB cable or powered using the AC-to-DC adapter or battery. Control is implemented using the SSC-32 protocol.

The Lynxmotion 5LA Robotic Arm robotic arm has a rotational base joint, a vertical shoulder joint, a vertical elbow joint, a vertical wrist joint, a rotational wrist joint, and a two-fingered end-effector (gripper). The base can rotate 360° horizontally, while other joints can rotate 180° vertically. The grip can do holding and putting action. The control is also implemented using the SSC-32 protocol.

The course takes a project-based approach to teaching robotics. The approach involves giving students a robot to assemble, adding sensors, and providing increasingly complex challenges. Students are left to work on their lab assignments in teams. Learning typically follows the following scenario: 1) A group of students are presented with a problem and learning materials required for solving it. 2) The students study learning materials and select/adopt appropriate solutions under the guidance of the teacher. 3) The students construct, model and deploy a robot to empirically validate the solution. 4) The students present their solution to other students and the teacher at workshop.

TABLE II. SUMMARY OF ROBOTICS PROJECTS.

Project topic	Robot	Language	No. of students
Line following 1	Lego Mindstorms NXT	NXT-G	4
Line following 2	Lego Mindstorms NXT	NXT-G	4
Obstacle evading using ultrasonic sensor	Lego Mindstorms NXT	NXT-G	4
Sub-total (NXT-G)			**12**
Symbol drawing	Arduino 4WD	MVPL	3
Lawn mowing algorithm	Arduino 4WD	MVPL	4
Sub-total (MVPL)			**7**
Control of robotic arm 1	Lynxmotion 5LA	Text-based	4
Control of robotic arm 2	Lynxmotion 5LA	Text-based	4
Sub-total (Text-based)			**8**

For the final project of the course, the students had to arrange into groups of at least three persons and complete a project with a selected robot. The use of visual programming language was not enforced but highly recommended; groups could choose any language to program their robot. Projects completed in the course are summarised in Table II. The analysis of students' choice of language for their projects shows that 44 % of students have selected Lego NXT-G as compared to 26 % of students that have selected MVPL, and 30 % of students – the text-based language (Arduino C).

IV. CONCLUSIONS

Three main problems of visual programming languages were identified: scalability, readability and speed of input.

The scalability problem arises as with increasing program size, visual languages become less usable and face the so-called *Deutsch Limit* [10], which states that '*you can't have more than 50 visual primitives on the screen at the same time*'. The solution of this problem requires more effective methods of using screen area (e.g., automatic layout) and going beyond the 2D representation to achieve more effective representation of program's structure without introducing unnecessary over-complexity.

The readability problem arises due to inherent parallelism in the robotics domain and complexity of input/output relations between blocks which lead to difficulties in understanding a program. Many students found visual languages harder to read and understand than textual ones, especially for larger programs. However, this is not sufficient to conclude that visual languages are inherently harder to read than textual. The already formed habits of the students could have made them biased. The problem can be addressed by using colours and other graphical effects (e.g., animation) more effectively beyond simple separation between groups of blocks to represent different views of a program.

The speed of input for visual programming languages still remains below that of textual languages using the keyboard, which may be a reason why professional programmers prefer text-based languages. The problem could be addressed by introducing more design automation and raising the level of abstraction above simple structures of structural programming (such as loops) to the idiom or pattern level.

The graphical representation of building blocks in MVPL and NXT-G is different. MVPL has minimalist design, with plain blocks and textual configuration parameters. NTX-G is colourful and represents almost all additional parameters by icons. This presents problems for both: MVPL is less visually appealing and friendly to novice users, while NXT-G might be considered both too difficult to understand and too childish to someone with programming experience. The NXT-G programs proved to be very large in real projects. Overall, students found programming with NXT-G more cumbersome than with textual programming languages (all students had previous programming experience).

Both MVPL and NXT-G examined in this paper lack identification and interaction capabilities which are needed to integrate a robot into the Internet of Things (IoT). They are aimed at controlling a single robot from a computer. To make these languages suitable for the IoT applications, other services providing robot identification, data exchange, interaction, and data sending to web services are needed.

REFERENCES

[1] B. Donald, J. Jennings, D. Rus, "Minimalism + Distribution = Supermodularity", *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2–3, pp. 293–321, 1997. [Online]. Available: http://dx.doi.org/10.1080/095281397147130

[2] C. Turcu, C. Turcu, V. Gaitan, "Merging the Internet of Things and Robotics", *Recent Researches in Circuits and Systems*, pp. 499–504, 2012.

[3] R. Burbaite, R. Damasevicius, V. Stuikys, "Using Robots as Learning Objects for Teaching Computer Science", in *Proc. of the 10th World Conf. on Computers in Education (WCCE 2013)*, Torun, Poland, 2013, to be published.

[4] B. A. Myers, "Visual programming, programming by example, and program visualization: a taxonomy", in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems* (*CHI 86). ACM*, New York, NY, USA, 1986, pp. 59–66.

[5] E. J. Golin, "A method for the specification and parsing of visual languages", Ph.D. dissertation, Brown University, 1990.

[6] J. J. Lee, J. Hammer, "Gamification in Education: What, How, Why Bother?", *Academic Exchange Quarterly*, vol. 15, no. 2, 2011.

[7] T. Green, M. Petre, "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework", *Visual Languages and Computing 7*, pp. 131–174, 1996. [Online]. Available: http://dx.doi.org/10.1006/jvlc.1996.0009

[8] E. V. Howard, "Visual Programming: Concepts and Implementations", M.S. thesis, Miami University, 1994.

[9] K. A. Nguyen, "A case study on the usability of NXT-G programming language", in *Proc. of 23rd Conf. in Psychology of Programming*, 2011.

[10] A. Begel, *LogoBlocks: A graphical programming language for interacting with the world (AUP)*, MIT Media Lab, 1996.