Robust - Towards the Design of an Effective Control Library for Lego Mindstorms NXT

Konrad Kułakowski

Institute of Automatics,
AGH University of Science and Technology
Al. Mickiewicza 30,
30-059 Cracow, Poland

Abstract. Is Lego Mindstorms NXT just a toy? Some argue that it is not. Unfortunately, if you look closer at Mindstorms it is easy to see many drawbacks, such as limited hardware resources or sensor impreciseness, which prevent people from implementing more complex systems for this platform.
This article presents the Robust library - middleware that tries to overcome some of these problems. A method for efficiently moving significant parts of the control algorithm to a PC computer, as well as improving the capacities of standard sensors and effectors by grouping them into complex devices are the technologies provided by the library.

1 Introduction

During the last decade we have observed the increasing popularity and accessibility of simple robotic constructions designed for young people interested in robotics. An example of such a product is *Lego Mindstorms NXT* - an educational toy which quickly stopped being just a toy [11] and became a didactic tool used in colleges and universities [2,18]. Although *Lego Mindstorms NXT* is popular among people who deal with intelligent control systems, they have to deal with several major drawbacks of the NXT platform. This article presents the architecture of the *Robust* middleware library. This tries to overcome some of the typical problems connected with NXT usage, such as limited hardware resources of the intelligent brick, inefficient bluetooth based communication between NXT and the PC, or the annoying shaky manner of moves caused by the think-drive-think-drive operation cycle. It is worth noting that this is probably one of the first solutions addressing communication problems between PC and NXT in the context of building of an intelligent control system.

Robust is designed as an object-oriented software and is implemented in Java - an object-oriented language. As the most suitable design language for this class of applications, UML 2.0 (*The Unified Modeling Language*) [17] has been adopted in all the design diagrams presented in this paper.

The first introductory part of the article contains a brief introduction to the *Lego Mindstorms NXT* platform and *Lejos*. It also contains the drawbacks, that justify the efforts connected with creation of the *Robust* library. The second section presents the essential parts of the *Robust* architecture. The final part provides the current status of the library and summarizes the paper.

1.1 Motivation

The central part of each *Lego Mindstorms NXT* set is a so-called intelligent brick - a control unit equipped with a 48MHz ARM based CPU, 256KB ROM and 64KB RAM memory, ports for plug sensors and effectors, a USB and BT (Bluetooth). Of course, such a configuration is not sufficient to handle complex control algorithms using a large knowledge base and world model. Thus, the natural way of overcoming the problem is to move the control algorithm from the NXT intelligent brick to a PC computer and control the brick remotely via BT. Unfortunately, it appears that the default control mechanisms provided by Lego firmware are not sufficient for assuring the reliable and robust operation of the system. In this approach every single reading of a sensor has to be passed on via a BT link, which makes the frequent sensor readings extremely ineffective. Tests performed by the author show that real-time observation of the states of many different sensors using default NXT firmware and BT based control libraries, such as ICommand, is almost impossible. The BT link is overloaded, some sensor readings come too late and others are lost.

The solution proposed by the author relies on providing the *Robust* control library which constitutes a kind of middleware for more complex control systems written in Java. It supports asynchronous communication for almost all standard sensors such as ultrasonic, sound, touch and light sensor and some third party sensors such as a magnetic compass or camera. Another important benefit of *Robust* is the concept of complex effectors and sensors. In other words, single effectors or sensors might be joined into a group, which is treated as a virtual hardware. Such a virtual hardware may offer the programmer larger capabilities than all of its parts individually. For instance, in the *Robust* library two motors that drive a standard vehicle built upon the NXT hardware, such as TriBOT [13,1] are joined into one virtual effector called *MoveGenerator*. Due to this combination, a robot controlled via the *Robust* library is able to eliminate jerking between moves (every move smoothly connects with the next one). Moreover, because a move might be queued in advance it is possible to avoid annoying, widely known in the literature [16] move-think-move-think operation cycles.

1.2 Lego Mindstorms NXT

Lego Mindstorms NXT was released in 2006 as a continuation of the widely recognized line of Lego educational toys. This is a set of Lego blocks, simple sensors, servomechanisms and a control unit (intelligent brick) powered by an ARM based processor. It allows the user to build a wide range of robotic models from standing ones, such as a robotic arm or elevator, to mobile ones, like the famous TriBOT, Spike and others [13,1,14,5] *Lego Mindstorms NXT* is provided together with the National Instruments LabVIEW Visual Programming Language that allows the user to create and run simple control algorithms. Since the hardware specification of the NXT intelligent brick is open, besides the original firmware and the LabVIEW programming environment, there are many third party firmwares providing the ability to program the *Mindstorms NXT* in languages such as Ada, C, C++, C#, Perl, Python, Visual Basic, Fort, Fortran, ObjectiveC or Java. It opens the NXT platform to a wide audience of programmers and brings about the chance to create more sophisticated control algorithms than is possible in the standard toolkit. The criticisms of *Lego Mindstorms* (besides the lack of advanced sensors and effectors) are focused on the limited hardware resources, such as RAM and ROM memory and the computational power of ARM based processor [10].

1.3 Lejos

Lejos is a *Java Virtual Machine (JVM)* available as a firmware replacement for the *Lego Mindstorms intelligent brick*. It provides the programmer limited, but still quite substantial, Java functionality including multi-threading, garbage collecting and standard data types and algorithms. Standard API is enhanced by the *lejos.** package, which provides access to the specific hardware resources such as motors, sensors, LCD screen or loudspeaker. The *Lejos* application has to be first cross compiled on a PC computer, then uploaded to the *Mindstorms NXT*. *Lejos JVM* provides a convenient menu, which allows users to run the uploaded application directly on the hardware. There are two popular architectural paradigms of the *Lejos* application: distributed, where the application logic is shared between the NXT intelligent brick and other devices (mobile phone, PC etc.) and non-distributed (concentrated) [3,15,4]. The *Robust* library takes its benefits from the distributed approach.

2 Robust library architecture

2.1 Architecture overview

The *Robust* library is composed of two modules *RobustNXT* and *RobustPC*. The first one runs directly on an intelligent brick, such as the *Lejos* application, whilst the second one is a ready-to-use Java library compatible with JVM (*Java Virtual Machine*) version 5.0 (fig. 1). Both parts of *Robust* communicate with each other via a BT link.

RobusPC provides to the programmers with several APIs *(Abstract Programmer Interfaces)* that allow them to successfully control many different hardware NXT components. There are two kinds of interfaces: synchronous and asynchronous. The asynchronous interfaces allow user to define an action listener object, which is triggered only if it is necessary. There are two administration interfaces: *RobustAPISystem* and *RobustAPISystemAsync*. They provide access to the methods controlling startup and shutdown of the library etc. For instance, *RobustAPISystemAsync* provides the possibility to define the handler responsible for logging debug messages which come from the intelligent brick.

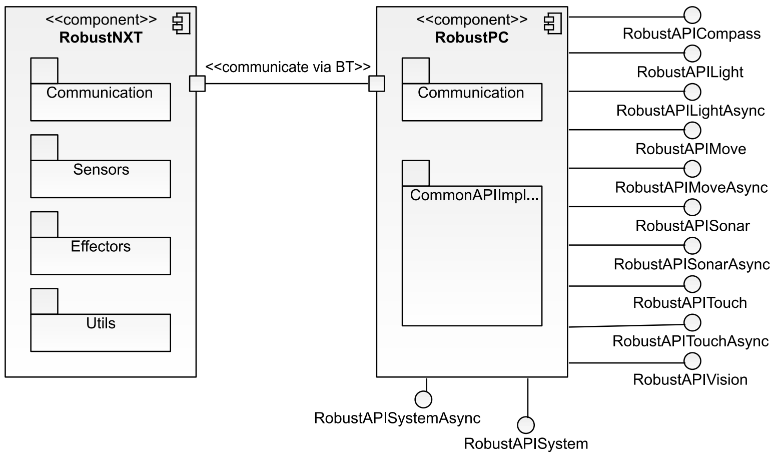


Fig. 1: General Robust Architecture

2.2 Communication

Both *Robust* components contain parts responsible for communication. They communicate with each other sending messages via BT. The message class *NXTBTMsg* (fig. 2) is serialized and deserialized at both ends of the connection.

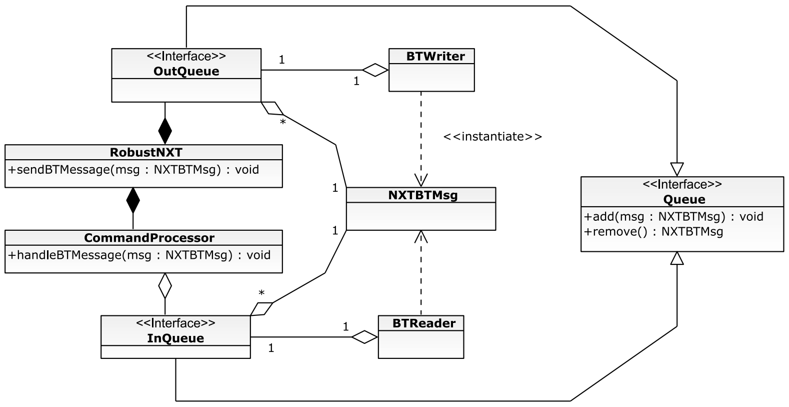


Fig. 2: Robust Communication Scheme

Prior to being serialized the message has to be put into the *OutQueue*. Next it is taken by the *BTWriter* thread and after serialization is sent out to the second module. *BTReader* is responsible for listening out for incoming messages. When the message comes it reads them from the BT stream and puts them to the

InQueue. Then the message is read by the *CommandProcessor* responsible for its further processing. Figure 2 shows the communication mechanism implemented in *RobustNXT,* however, in the *RobustPC* the scheme is very similar.

2.3 Asynchronous communication, complex sensors

Implementation of some parts of the system directly on the intelligent brick creates the possibility of shortening the communication distance between the system and hardware. It results in the ability of to provide asynchronous communication interfaces for all the sensors available on the *Mindstorms NXT* platform. An example of such a solution might be the *Robust's* way of handling of an ultrasonic sensor (fig. 3). It is composed of a few classes spread out among both modules *RobustNXT* and *RobustPC*.

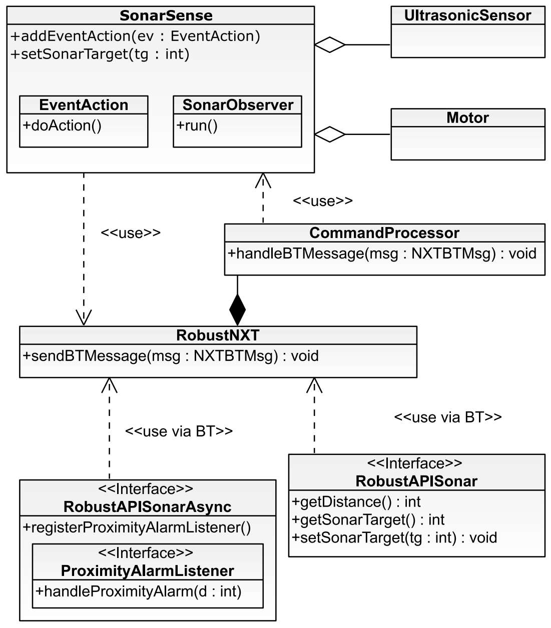


Fig. 3: Example of sonar sensor design

In *RobustPC* the ultrasonic sensor functionality is available through two interfaces - *RobustAPISonarAsync* and *RobustAPISonar*. The first one provides a method which allows user to register a proximity alarm listener. The listener object has to be inherited from the interface *ProximityAlarmListener* defined inside *RobustAPISonarAsync*. Implementation of the *handleProximityAlarm()* method allows asynchronous processing of sonar events.

The second interface *RobustAPISonar* provides three methods, where one of them allows the user to measure the distance to the obstacle. Since sonar

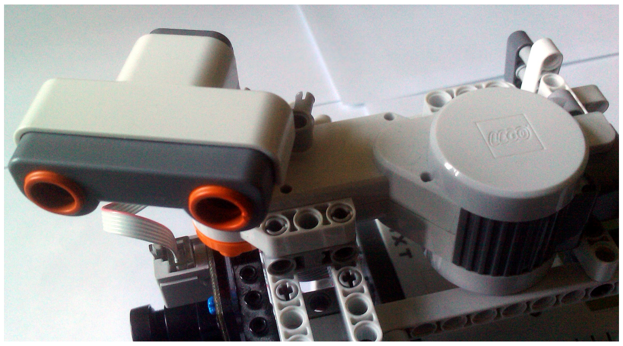


Fig. 4: Complex sensor - ultrasonic sensor and motor

is designed as a complex sensor, it is composed of an ultrasonic device and servomechanism, where the effector is responsible for the direction of distance measurement (fig. 4). Thus, the remaining two methods in *RobustAPISonar* allow the user to get and set the measurement direction, called in this context the target.

In the second module, *RobustNXT*, handling of both asynchronous and synchronous APIs converges into one class *SonarSense*. On one hand it aggregates objects representing the actual ultrasonic sensor and motor, on the other hand, it provides a mechanism for tracking state changes of the sonar. If the state change matches the specification provided by the subscriber, the appropriate message is sent out back to the *RobustPC* module. The whole notification mechanism is a modified version of the well-known design pattern *Observer* [6].

2.4 Complex effectors

Like sensors, effectors can also be combined into groups - complex effectors. Due to the fact that such a complex effector is partially implemented in *RobustNXT*, mechanisms that form the effector group might be better synchronized.

The *Robust* library provides two implementations of complex effector *MoveGenerator* (fig. 5). The first one *MoveGeneratorB* (B as Basic) is a combination of two servomechanisms that are used for propelling constructions based on the standard *Lego Mindstorms NXT* vehicle *TriBot* [13]. *MoveGenerator* allows the robot to perform three types of move:

– move forward - both wheels rotate at the same velocity and in the same direction (fig. 6a).
– turn left or right smoothly - both wheels rotate in the same direction but with the different velocity (fig. 6b).
– turn left or right "in place" - both wheels rotate at the same velocity but in the different directions (fig. 6c).

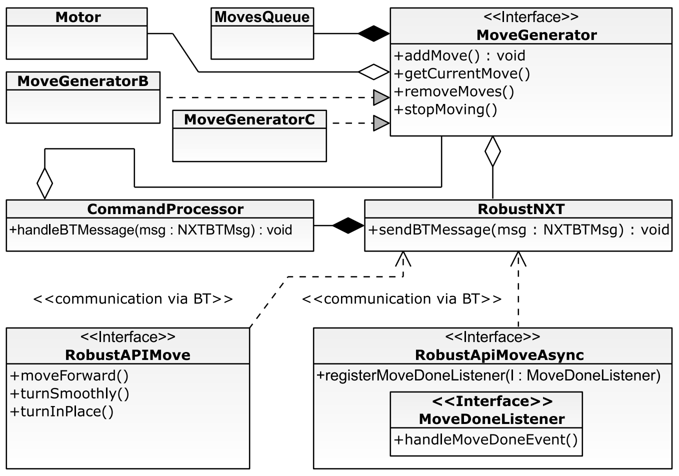
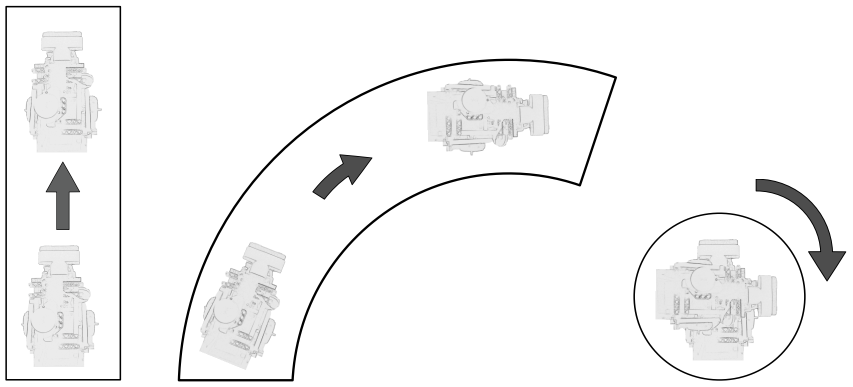


Fig. 5: Move generator

Interface *RobustAPIMove* enables the move command to be sent to the *Robust-NXT*. All the moves can be parametrized, so move length, velocity, turn degree and radius can be easily adjusted to the actual control needs.



(a) Forward move (b) "Smooth" turn (c) "In place" turn

Fig. 6: Move primitives

The sent move command is queued into *MovesQueue*, then it is pulled out for execution by the controller . Thus, apart from the execution of the very first move primitive, the system knows the next move primitive before the execution of the current move primitive is completed. This fact allows it to go from one move to another smoothly, and it helps to avoid annoying jerking between different move primitives.

Increasing of the precision of moves, as well as minimizing the errors of odometric readings is still a challenge for researches [21,8,20,19]. Thus, in order to increase the move precision another *MoveGenerator* (*MoveGeneratorC*) has been implemented. It is able to control the track of the vehicle using the motors' odometric readings as well as the indication of the magnetic compass. Unfortunately, the performed experiments show that in the standard office environment there are many sources of magnetic distortion. Thus, *MoveGeneratorC* is efficient but only in specific environment, where there are no metal items, wires with current etc.

Move generator also offers the asynchronous control interface *RobustAPI-MoveAsync* (fig. 5). Since move primitives are queued and executed in turn, the *RobustAPIMoveAsync* allows users to subscribe for the event, which is informing that the new move primitive is about to be processed.

3 Summary

It is hard to classify *Lego Mindstorms NXT* as only a toy for smart children [11]. There are a lot of examples of *Mindstorms NXT* pretending to be much more than a toy [18,9,7]. On the other hand, it suffers from problems that are typical for amateur constructions (or just simply toys) such as impreciseness of sensors, very limited hardware resources or limited choice of components. The *Robust* library has been designed and implemented by the author in response to problems identified during experiments with *Lego Mindstorms NXT*. The problem of limited resources that are usually insufficient for running complex control algorithms is overcome by moving a significant part of the control system to a PC class computer. The imperfections of standard components can by reduced by practical use of the concept of complex sensors and effectors.

Of course, not all the problems can be solved by the use of software. Sensor readings might be improved, but not eliminated. The impreciseness of moves resulting from the clearance of blocks may be estimated but not avoided. Despite all the unsolved issues, the preliminary results are very promising. The experiments performed prove the usefulness of the *Robust* library in standard problems such as searching or localization [21].

The *Robust* library is still being developed. Its early version can be downloaded from sourceforge.net [12].

Acknowledgment

This research is supported by the AGH UST, contract no.: 10.10.120.105

References

1. *Lego Mindstorms User Guide.* The Lego Group, 2006.

2. S. Atmatzidou et al. The use of LEGO Mindstorms in elementary and secondary education: game as a way of triggering learning. In *In Workshop proceedings of International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2008.
3. B. Bagnall. *Maximum LEGO NXT: Building Robots with Java Brains.* Variant Press, 2009.
4. G. Ferrari et al. *Programming LEGO Mindstorms with Java.* Syngress Publishing, Inc., 2002.
5. M. Ferrari, G. Ferrari, and D. Astolfo. *Building Robots with LEGO Mindstorms NXT.* Syngress Media, 2007.
6. E. Gamma, R. Helm, R Vlissides, and R. E. Johnson. *Design Patterns.* Addison-Wesley, 1997.
7. B. S. Heck, N. Scott, and Ferri. A LEGO Expermment for Embedded Control System Design. *IEEE Control Systems Magazine*, 2004.
8. J. M. Holland. *Designing Autonomous Mobile Robots: Inside the Mind of an Intelligent Machine.* Newnes, 2003.
9. T. K. Iversen and et al. Model-checking real-time control programs: verifying LEGO Mindstorms systems using UPPAAL. *IEEE Computer Society*, 2000.
10. F. Klassner. A Case Study of LEGO Mindstorms 'TM Suitability for Artificial Intelligence and Robotics Courses at the College Level . In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.
11. F. Klassner and S. D. Anderson. LEGO MindStorms: not just for K-12 anymore. *Robotics and Automation Magazine, IEEE*, 2003.
12. K. Kułakowski. http://sourceforge.net/projects/robust/.
13. Lego Mindstorms. http://mindstorms.lego.com/.
14. A. end others Miguel. *LEGO Mindstorms Masterpieces - Building and Programming Advanced Robots.* Syngress Publishing, Inc., 2003.
15. J. A. B. Moral. Develop Lejos programs step by step.
16. N. Nilsson. Shakey the Robot. Tech Note 323, AI Center, SRI International, 1984.
17. OMG. Unified modeling language (uml) version 1.5. Technical report, OMG, 2003.
18. S. Parson and E. Sklar. Teaching ai using lego mindstorms. *American Association for Artificial Intelligence*, 2004.
19. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 2000.
20. S. Thrun. *Robotic Mapping: Survey*, chapter 1. Morgan Kaufmann, 2003.
21. S. Thrun, W. Burgard, and D. Fox. *Probabilitc Robotics.* MIT Press, 2006.