Boot The Bot: Java-based Simulation, Code Generator and Live Controller for ABB Robots

Richard Dank
Institute of Architecture and Media, Graz University of Technology, Austria
Email: dank@tugraz.at

***Abstract*—Half a century has passed since the introduction of the first Unimate. Now robots finally have arrived in the minds (and projects) of creatives worldwide. This paper discusses issues concerning algorithmic and innovative offline programming of ABB robots in the context of artistic and architectural purposes. It compares existing software tools for generating RAPID code and introduces *Boot The Bot* – a standalone, cross-platform digital tool to generate or import points plus extra data and export valid RAPID code for immediate execution on ABB robots. This program is the basic software application for several projects and subsequent research themes in TU Graz's field of expertise *Resource-Efficient Non-Standard Structures*. And it can operate live and online.**

***Index Terms*—project and practical application of Algorithmic Design, Digital and physical robotic interfaces, Robotics in art and architecture, 1:1 production, Java/Processing.**

I. Introduction

While harmonized program code gained currency for regular CNC systems, a standard for robot code could not be implemented yet.

Plotters, the very first computer numerical controlled (CNC) machines that made it out of the factory into the offices, basically run on vector graphics markup languages. One of those derivates originally used on Gerber Scientific plotters and further developed and adapted at the MIT Servomechanisms Laboratory in the 50s today is the de facto standard for CNC programming languages: G-Code. This method is so widely used that it was accepted early by the Electronic Industries Alliance (EIA), then incorporated by the Deutsche Institut für Normung into DIN 66025 and by the International Organization for Standardization into ISO 6983. Though most manufacturers require a slightly different syntax they all rest upon G programming language. One could say that they just speak different dialects.

As mentioned before, robots do not offer such convenience. The big companies use completely different proprietary programming languages. Industrial robots produced by Unimation Inc. (now owned by Stäubli[1] [1]) for example rely on the Variable Assembly Language (VAL), KUKA[2] on KUKA Robot Language (KRL) and ABB[3] machinery is controlled with RAPID Code. [2] They fundamentally differentiate from each other – not only in terms of semiotics, semantics and syntactics. The interpreters require entirely different datasets to be able to move the robotic arm. This leads to several advantages as well as drawbacks.

In the following paragraphs I would like to quickly sum up these characteristics of basic motion instructions and data types for ABB robots, so one could grasp the challenges there. Later I would like to present three other superb applications for RAPID offline programming, to be able to assess *Boot The Bot* (BTB). In the end the text shall outline some of the projects designers and artists have made utilizing BTB software and ABB machines.

II. ABB RAPID Code: Tools, Work Objects, Targets and Instruction Sets

Of course there are quite a few motion instructions, still I want to explain only the seemingly three most important of them concerning ABB six axes robots.

MoveAbsJ directly rotates the six axes of the robot to a distinct joint position – very common at the beginning and end of any sequence or for 'untangling' the robot's arm. The instruction below makes the tool0 (the flange itself) move along a non-linear path to the absolute axes positions (prior stored by user definition in the joint target) jPos_0000, with the default speed data v250 (velocity) and zone data fine (precision).

MoveAbsJ Joints_0000 , v250 , z100 , tool0 ;

MoveJ is used to move the robot tool to a certain position on the work object not using a straight line – typically used for picking and placing things. In practice this means without simulation one cannot predict the movement of the machine. All constants (Target_0000, weldingPen and synthMat0) must be predefined, so that the data is available when calling the instruction.

On the other hand MoveJ avoids any problems with the rotation of one of the six robot axes (Fig. 1, left). Plus you don't run into singularity troubles that occur when two robot axes come close to being aligned.

Manuscript received October 24, 2012; revised December 10, 2012.

Fig. 1., 2., 3. and 5. adapted graphics by ABB. Fig. 4. copyright Richard Dank. Fig. 6. and 7. photographs by Philipp Sackl. Fig. 8. photographs by Konstantinos Tzivanopoulos and Thomas Raggam.

[1] http://www.staubli.com/

[2] http://www.kuka-robotics.com/

MoveJ Target_0000 , v200 , z0 , weldingPen\WObj:=synthMat0 ;

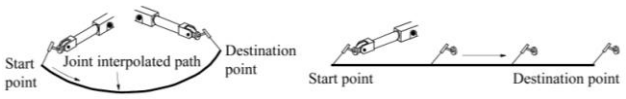

Figure 1. MoveJ is a fast and robust but mostly unpredictable way to move between two points (left). Whereas MoveL describes a linear path, it can run in a number of problems – e.g. the start point and the destination point are too far apart (right). [3]

Finally *MoveL* is used to move the tip of the tool linearly to a given destination (Target_0001). It is widely used for most operations, still it poses major risk to executable RAPID code under certain conditions (Fig. 1, right).

MoveL could result in singularities (Fig. 2). But what's more, when the robot needs to pass distant targets or change the orientation of the tool a lot, the software controller of the robot is not able to carry out the task, as none of the six axes can be turned more than 90 degrees within a single instruction.

MoveL Target_0001 , v100 , z5 , weldingPen\WObj:=synthMat0 ;

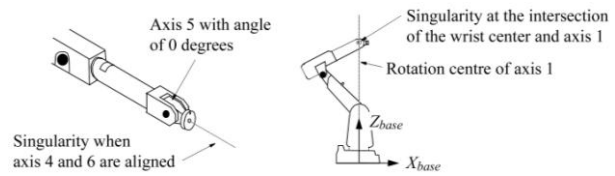

Figure 2. Wrist singularities occur when axis 5 is 0 degrees (left). An arm singularity where the wrist center and axis 1 intersect (right). [3]

The definition of the *robtargets* is tricky as well. The first set of triple values in box brackets are the X-, Y- and Z-coordinates of the tool tip. The second four values form the orientation of the tool in quaternions, a complex notation of three-dimensional rotations. Both must be calculated in relation to the work object the target later is assigned to. The last list (9E9,9E9,9E9,9E9,9E9,9E9) is reserved for controlling additional axes like linear tracks. In the code excerpt beneath there is actually none.

However, the third array indeed requires most of the attention. [1,1,-1,1], for example, defines the configuration of the robot. Considering axis 4 and 6 of the robotic arm, if the tool is able to reach a target at all (Fig. 3, left), it can be done in at least four different configurations. But depending on the model specification and the position/rotation of the target the number of possibilities increases dramatically (Fig. 3, right). And ABB robots require valid configuration, otherwise the exported code won't work at all.

Although the data types for the tool (*tooldata*) and the work object (*wobjdata*) contain additional information, the definition is quite similar to the robot targets. But all details shall not be delineated here. For further information see e.g. the technical reference manual for the RAPID programmer from ABB Robotics Products [4].

CONST robtarget Target_0000 := [[257.95,474.5,0.0] , [0.060166642,0.010343712,-0.9836818,0.16924272] , [1,1,-1,1] , [9E9,9E9,9E9,9E9,9E9,9E9]] ;

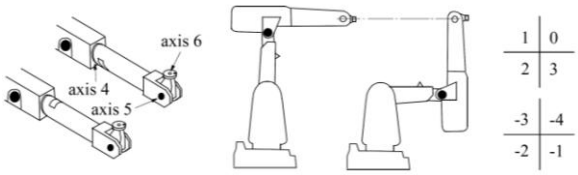

Figure 3. Two different wrist (left) and arm configurations to attain the same position and orientation (center). Quarter revolutions for the joint angles as final target configuration (right). [3]

III. Offline Programming in RobotStudio, Pi-Path and Hal

RobotStudio of course is ABB's very own Software product. It gives you full access to on- and offline programming. This includes the generating of complete programs for example from any linear motion data you import. Apart from that it is 'the' perfect simulation tool for any RAPID code as it is built on the VirtualController, "an exact copy of the real software that runs your robots in production."[4]

With all the high performance it still has some shortcomings. First and most important of all it is closed source with no scripting environment. This means everything has to be done manually. Evidently this means that it is by definition not capable of "customized scripting" – "fluent bottom-up architectural design […] workflow" [5]. And it is not intended to be.

Secondly, once you have a tool path and the position/orientation of the targets – which is of course easy to calculate anyway - the automatic definition of the robot targets is rather slow. RobotStudio needs to completely simulate the robotic arm's motion in order to find out any problems with the configuration. This is very accurate, and it should be! But if anything unexpected happens – remember the singularity- and rotation-troubles outlined in the prior chapter – the user must again solve this by hand. Plus if there are thousands of targets, it might not be able to compute that number of instructions after all.

As a result RobotStudio is a perfect tool for a small number of, especially taught, targets and as a final simulator before going into production.

Master Automation Group's (MAG)[5] *Pi-Path* should be briefly mentioned here as an easy yet powerful tool to convert three or five axes CNC code into multi-axes robot programs. As it is basically meant to translate pre-computed APT-files, it needs manual input on errors as well. It is ultra-fast in writing the configuration data, but again can not be programmed. It is perfect for milling procedures, can be 'abused' for other tasks, but is not able to parse elaborate code like different motion instructions etc.

[3] http://www.abb.com/robotics/

[4] http://www.abb.com/product/us/9AAC111580.aspx

[5] http://www.mag.fi/

Finally we want to take a look at *HAL*. Alike robotsinarchitecture's Parametric Robot Control (PRC)[6] for KUKA, HAL is a plug-in for Rhino's[7] generative modeler Grasshopper[8]. It was developed by Thibault Schwartz[9] in 2011 and is revised ever since. The current unlicensed version 0.03 can simulate ABB robots, whereas the full version is also able to export RAPID code. Just like so many Grasshopper extension out there today, it is an easy to implement way to program ABB industrial robots. It contains a large library of components, including default objects for wire cutting and milling. And it is fast. Backed by the vast potential of Rhino it can do almost everything one might call for. Nevertheless we want to take a look at what it cannot do – at least yet.

As mentioned earlier robots just move from one target to the next, which poses a problem for several operations. HAL produces code for points well that are close to each other, but it doesn't take configuration problems or singularities into account – unless you solve them manually. Apart from that it assumes that there is only one viable configuration for one target, which again could make things harder.

To sum it up, HAL is sophisticated enough for ingenious applications, still straightforward so that even inexperienced users can work with it. Definitely a recommendation for everyone that works with Grasshopper – though this means being chained to Windows as well.

IV. Boot the Bot

So after these observations on other software packages, why should there be room for another fish in the pond?

Soon after we at the Graz University of Technology (TUG)[10] decided to acquire ABB robots in 2009/10 it became clear that we needed to directly control the machines in order to be able to produce remarkable output. At that time HAL was not around yet. Otherwise we might have considered enhancing the release and making it suitable for our demands. On the other hand RobotStudio and Pi-Path are not versatile enough and can only be deployed as post-processors. Moreover we agreed that, in the long run, it is valuable and rewarding to be able to understand and manipulate a machine of that capability on an advanced level – especially for architects and artists. We needed to understand the tool we were using as thoroughly as possible. So we – Jacob Wegerer and I – started to develop our own kinematic solver: *Boot The Bot* (Fig. 4).

We wanted the application to be as open as possible. Thus it is written under the creative common license in Java[11] with the implemented integration into Processing[12] – everything open source. We deliberately did not fall back on a CAD backbone like Maya/MEL [13] or Rhino/Grasshopper and of course we had to take a loss there. Designers that are not willing and/or able to calculate the initial points of the motion paths in 3D will not be able to use the full potential. But as all architecture students at the TUG already have the mandatory class Digitale Methoden der Gestaltung (DM2) in their third semester, we accepted that. In DM2 students are obliged to take their first steps into contemporary algorithmic design methods. In essence we teach them the basics of programming there [6].

Figure 4. Boot The Bot with the help window faded in: Left the interactive visual surface, right the console output.

Thus we were able to open BTB to all software platforms. Moreover, by using Java we made it possible that projects could easily access live and/or online input. Another interesting aspect of BTB is that data, like the measured work objects or tools on the real robot, can be read and interpreted. So you can make declarations both ways – either generate them or grab existing ones.

However the key feature is the automatic configuration of the targets. Multiple possible configurations are calculated and due to several parameters BTB then decides which is best. If there is still no satisfying solution, the motion path is interpolated and additional crucial targets are added. On the other hand a general interpolation of linear movement would lead to a lot of unnecessary lines of code. Depending on the number of instructions the robot should execute, this is vital. We don't want to run into memory overflow on the machine.

By the way, all potential configurations can be set manually as well. Just like in RobotStudio. Even singularities can be avoided to a certain extend. If possible the robot just moves around them.

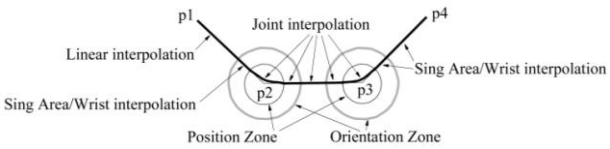

Figure 5. A fully working, yet complex motion path can be programmed manually like in the case above [3] or automatically generated with BTB.

As we didn't want to address programmers only, there are several connections for interchanging data. BTB can import formats like 3DM from Rhinoceros 3D, Autodesk's closed source DWG and the widely implemented Drawing Interchange File Format DXF. Thus BTB can be linked to virtually any existing CAD software out there. Furthermore it can open and save a specially designed plain-text exchange format. So any scripting language can generate the basic coordinates –

[6] http://www.robotsinarchitecture.org/kuka-prc/
[7] http://www.rhino3d.com/
[8] http://www.grasshopper3d.com/
[9] http://thibaultschwartz.com/
[10] http://www.tugraz.at/
[11] http://www.java.com/
[12] http://processing.org/
[13] http://autodesk.com/maya/

plus additional target data if desired –, pass it on and use BTB just as a kinematic solver, simulation engine and RAPID code generator.

Boot The Bot has several displaying and console output modes implemented by default. It can be toggled between orthographic and perspective view. Plus if delivers a large set of standard and randomly generated motion paths. All of that is very valuable for setting up the real robot, the tool, the work object and for general debugging.

But most of all it is designed for live input connections and as programming extension for Java-based applications. This means that there is e.g. an especially integrated patch for the Processing sketchbook where to write your own code that then generates the targets. Consequently these figures can originate from milling paths, image data, some type of interaction or any other kind of algorithmically converted information. It is completely up to the user's imagination and creativity.

In summary, there are two ways of utilizing BTB. First of all one can just take it as a multi-platform post-processor for converting target information into RAPID code to address ABB robots. The environment allows for

- Importing a variety of CAD-formats and plain-text data delineating motion paths
- Different visual and textual screen displaying modes, orthographic and perspective views for debugging
- Additional positioning, motion and timing settings
- Tool and work object import and export
- Full simulation of the complete workflow
- Export working ABB RAPID code.
- Direct physical control over the robot via ftp connection.

Nonetheless, the intrinsic intention of BTB is the standalone version. It covers all the features above but additionally enables the user to

- Compute the robot's motion by self-written functions completely within BTB .
- Grab live input and/or interact with the output.
- Generate (convert data), verify (simulate the sequence) and finalize (write RAPID code) all in one non-linear procedure.
- Run it locally or on the web.

And again I need to mention the fact that everything is open source. So if someone is not happy with the automatic configuration parameters, the interpolation steps, the interpretation of fragmentary motion data or even the colors

- Everyone can adapt the code for their own needs.

V. Projects using BTB

But of course only the implemented projects bring such software to life – at least from the point of view of the creatives. The current chapter will exemplarily present some significant works that built upon BTB. Obvious areas of operations – like milling and wire-cutting – shall not be referenced here. I will focus on standalone, interactive experiments at the transition between art and architecture. Most of them originated in the vicinity of the Design Master Studio *papier peint*[14] – conducted by Richard Dank, Christian Freißling and Urs Hirschberg at the Institute of Architecture and Media (IAM)[15]. The leading idea was "based on Semper's notion that it is the surface of walls that most strongly influences our special perception." It "was about developing surface effects and their prototypical production with the help of robots, with the aim of redefining a precise architectural setting. According to their designs, students had to write their own programs, […] construct the proper tools for the robot, and finally have the robot […] turn them into physical reality 1:1 scale." [7]

For example *motionMATRIX* by Marvi Basha tracks traces of a stair-climbing human body and then superimposes the robot's joint rotations. The resulting curves were then realized with an UV-color-pen in the hallway up to TUG's Zeichensaal 3 and 4 (Fig. 6, left).

Kathrin Hiebler's *Raumverfremdung* is a virtual dissolution of a roof truss corner. An algorithmic, anamorphic distortion and expansion of the purlin construction was processed into a grayscale developed view and realized with a rotating stamper. So depending on the distance to the artwork one perceives the imprints of varying intensity or catches sight of an infinite network of random wooden beams (Fig. 6, center).

Stefan Höll took papier peint – French for painted paper or simply tapestry – literally. He programmed *Individual Wallpaper* where one could sketch on a graphic tablet. Depending on the pressure and drafting speed the image was transmuted into line patterns (Fig. 6, right). The predefined wall geometry then led to automatic tiling for the production with different pens on paper-rolls that were transported into place by a customized, Arduino-controlled[16] conveyor table.

Figure 6. Project posters by Basha and Hiebler, final Image of Raumverfremdung and Höll's Individual Wallpaper manipulated on a Wacom board17 (from left to right).

Peter Kaufmann and Robert Schmid decided to parametrically bleach two diagonally opposite standing couches. One of them shows the vestige of the authors sitting, the other the logo of their drawing studio – a stag with L-system-horns (Fig. 7, left). They put together a dye dripping infusion bottle tool for the ABB robot to carry out the intervention *Die Anwesenheit der Abwesenden und Hirsch*.

Simone Mayr used self-invented font to brush-paint enamel on thermoplastic road marking. *Ohne Worte* was designed as a flame-scarfed guidance system for a large architectural office, where the blank spaces of a

[14] http://iam.tugraz.at/studio/w10/
[15] http://iam.tugraz.at/
[16] http://arduino.cc/
[17] http://www.wacom.com/

theoretical text unveiled the names of the senior partners (Fig. 7, right).

[SYN]these is the attempt to make sound graphically tangible and legible through netlike diagrams. Paul Pritz rendered characteristic values of the assigned compositions with a pen-squeezing semi-fluid paint dispenser on sound absorbing dungaree for a rehearsal room (Fig. 7, center).

By the way, all these projects, and more using BTB, can the watched in action on IAM's Vimeo site[18]. "The […] hereby presented design- and research-approach is […] always based on the possibilities and constrains of the actual materialization, as its properties and the scope of fluctuation of its variables are embedded into the computer-based generative processes." A quote I borrowed from Achim Menges [8].

Figure 7. Kaufmann/Schmid's bleached deer on a couch, their and Pritz's tool, extracts from the robot-produced [SYN]these and Mayr's guidance system (from left to right).

Finally I want to quickly present the *China-ink Painting Robot*, because the full capacity of Boot The Bot can be perfectly depicted there. In 2011 a precursor was shown at the 200 year anniversary[19] of TUG. There visitors were invited to be sketched live in silver on glossy black cardboard by an IRB 140. They could sit down in front of a regular webcam, switch between different drawing styles and interact via mouse and screen to finalize their portrait. When satisfied, the application computed the motion paths, generated the RAPID code and copied the program for batch processing to the robot (Fig. 8, left). A few minutes later they could take home a piece of art.

Figure 8. Two Pictures from TUG's 200 years anniversary, the China-ink Painting Robot plus Wolfgang Tschapeller, Peter Cook and Marjan Colletti picking up their portraits at the HDA (from left to right).

At the exhibition *By all means – analogue/digital experimental settings*[20] in 2012 the robot painted texts, pictures of the exhibition and the visitors onto the glass façade of the House of Architecture (HDA) (Fig. 8, center). It was a more leisurely approach, as the Chinese ink and the vertical surface required a much slower operating procedure. So the process was redesigned and the performative aspects were highlighted for the exhibit. Finally the robots action and the imperfectly predictable behavior of the brush interacting with the transparent paper mounted on the glass background concluded in a surprisingly tense blending of the analog and the digital (Fig. 8, right).

[18] http://vimeo.com/ioiiii
[19] http://portal.tugraz.at/portal/page/portal/TU_Graz/2011
[20] http://www.hda-graz.at/event.php?item=6961&lang_id=en

VI. Conclusion and Outlook

"Today, innovative projects typically result from the collaboration of multidisciplinary teams that join forces from the initial design stages on." [9] That requires not only proficient teamwork but in this day and age a seamless incorporation of a wide range of tasks preferably in a single, open-source, platform independent application as well.

BTB grew for almost three years now. And it fulfilled the needs of most of the demands we've ran into when operating ABB robots. The current version is 2.8, but still there are optimizations to be made – namely in the range of the quaternion normalization, the final robot and tool configuration. And of course colleagues at TUG and project partners elsewhere are constantly requesting new facilities and functionalities that we wish we would already have. As a consequence we keep customizing and developing Boot The Bot for new challenges – at least occasionally.

"An understanding of digital design as a unique set of design logic demands a formulation of the symbiosis between the product of design and the way it is now conceived, generated and materialized in digital media." Reframing Oxman's [10] statement, BTB provides this conjunction between the most open way of formalizing an idea – alphanumeric code – and the most versatile machine available – the robot.

Acknowledgment

The evolution of Boot The Bot started with stirring stimulus at IAM and energetic support of ABB – especially by Michael Roth. Apart from him Richard Dank certainly needs to give thanks to Jacob Wegerer. His contribution and collaboration exceeded his engagement as teaching assistant by far. And finally he would like to lay great stress upon the inspiring efforts and wonderful projects of all the peers and students that used BTB ever since it came out.

References

[1] G. Munson, "The Rise and Fall of Unimation Inc.," *Robot*, no. 24, pp. 36-41, September/October 2010.
[2] M. Wenz, "Automatische Konfiguration der Bewegungssteuerung von Industrierobotern," *Logos*, Berlin, pp. 95, 2008.
[3] ABB Robotics Products (ed.), *RAPID Reference Manual*, Västerås, 1998.
[4] ABB Robotics Products (ed.), *Technical reference manual: RAPID Instructions, Functions and Data types*, Revision J, Västerås, 2010.
[5] J. Braumann and S. Brell-Cokcan "Parametric Robot Control: Integrated CAD/CAM for Architectural Design", in *Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture*, Calgary, 2011, pp. 242-251.

[6] R. Dank "Why MEL," in *Proceedings of the 1st International Symposium on Algorithmic Design for Architecture and Urban Design*, ALGODE, Tokyo, 2011.
[7] R. Dank and U. Hirschberg, "Papier Peint," *GAM: Dense Cities*, no. 8, Springer, Wien/New York City, pp. 320-321, 2012
[8] A. Menges, "Uncomplicated Complexity," *GAM: Nonstandard Structures*, no. 6, Springer, Wien/New York City, pp. 140-151, 2009.
[9] M. Fellendorf and U. Hirschberg, "Design and Construction Science," *Forschungsjournal der TU Graz: Fields of Expertise*, special issue, pp. 16-21, 2008.
[10] R. Oxman, "A challenge for digital design and design pedagogy: theory, knowledge, models and medium," *Design Studies*, vol. 29, no. 2, pp. 99-120, 2008.

Richard Dank's main area of activity - scientifically and in practice - is the interweaving of art and architecture with/in the field of digital media. Collaborating with several architects, artists and institutions worldwide his oeuvre oscillates between designing and design - from graphics and videos, over installations and web applications, to construction and building -, always with the focus on algorithmic and interactive projects on- and offline. In the course of these encounters intense working residences and study journeys to America, Asia and all over Europe have been realized. Dipl.Ing. Dank received his Diplomingenieur (Master of Science) in architecture from the Graz University of Technology in 2006 and is currently working independently as richdank.com, moreover as a partner in the architecture cluster 0704 and teaching/researching as fulltime Universitaetsassistent (assistant professor) at the Institute of Architecture and Media.