MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications

Christina Deatcu*, Birger Freymann, Artur Schmidt, Thorsten Pawletta

Hochschule Wismar – University of Applied Sciences: Technology, Business and Design,
Research Group Computational Engineering and Automation, Philipp-Müller-Straße 14, 23966 Wismar, Germany
*christina.deatcu@hs-wismar.de

Simulation Notes Europe SNE 25(2), 2015, 69 - 78
DOI: 10.11128/sne.25.tn. 10293
Received: July 10, 2015 (Selected ASIM STS 2015 Postconf. Publ.); Accepted: July 20, 2015;

Abstract. Industrial robots are used in various fields of application and many robot manufacturers are active in the market. In most cases, their software solutions are proprietary and, consequently, they cannot be used for third party robots. Moreover, the integration of external hard- or software is highly restricted. Long term standardization efforts for robot programming languages, such as the Industrial Robot Language (IRL) and its successor, the Programming Language for Robots (PLR), have been mostly ignored by robot manufacturers. This fact leads to a restriction on the combined usage of robots. Multi-robot applications where robots have to interact are usually limited to software solutions and robots of one manufacturer. On the other hand, control design in engineering is often carried out by the usage of Scientific and Technical Computing Environments (SCEs) like MATLAB. The Robotic Control & Visualization Toolbox (RCV Tbx) for MATLAB/Simulink tries to close the gap between robot manufacturer-specific software solutions and SCEs. The current version of the RCV Tbx supports a uniform and integrated control development for KUKA and KAWASAKI robots in the MATLAB/Simulink environment. An extension to other robot types is straight forward. Thus, the implementation of heterogeneous multi-robot applications is considerably simplified.

Introduction

This paper is an extended version of [1] and aims to introduce the RCV Tbx for MATLAB/Simulink as an easy to use Rapid Control Prototyping (RCP) Tool for multivendor robot controls. The RCV Tbx has been under development by the research group *Computational Engineering and Automation* (CEA) at Wismar University since 2009 [2].

As research into robotics is proceeding rapidly and new fields of application for robots are being made up continually, the requirements concerning robot control development are increasing, too. Fast and easy control programming, integration of external hardware or software components and multi-robot operation are of particular importance. In this context it is often desirable to use a SCE, such as MATLAB, for RCP. RCP, according to Abel and Bolling [3], is understood as an integrated, continuous control development from early design to operating phase in a homogenous environment.

In addition to multi-robot capability, multivendor applications are one further key aspect. Today, various robot manufacturers are established on the market. They offer proprietary software environments with special robot programming languages such as KRL (VEN KUKA Robotics), AS (VEN Kawasaki Robotics) or RAPID (VEN ABB Robotics). From a software engineering point of view, all these robot languages are pretty similar. Nevertheless, long-term standardization efforts like the IRL and its successor, the PLR are still unsuccessful. In addition, almost all robot manufacturers offer a Computer Aided Robotic (CAR) system, which is also referred to as 3D robot simulation software. CAR systems typically provide physics-based robot models for one manufacturer as well as interfaces to 3D CAD systems. Thus, a simulation and 3D visualization of complete robot cells is supported. Robot controls can be developed within these virtual environments using the proprietary robot languages. Such CAR systems simplify the development and commissioning of robot applications. However, the proprietary software limits applications to products from its manufacturer. There are some third-party CAR systems available, such as 3DRealize-R by Visual Components Corporation [4]. They offer a comprehensive solution for simulation and 3D visualization of heterogeneous robot types.

However, the control programming is still based on the different proprietary robot languages. Hence, the development of interacting multi-robot controls is complicated for robots from different manufacturers.

For MATLAB/Simulink, besides the RCV Tbx [2], there exists a robot control toolbox for KUKA robots (KUKA Control Toolbox, KCT) developed at the University of Sienna [5, 6]. The KCT connects a remote MATLAB computer via TCP/IP to the robot controller of a KUKA robot. Usage of KCT is limited to one single robot from one manufacturer, namely KUKA, so that multi-robot and multivendor applications cannot be addressed.

Inspired by the idea of RCP in control theory, the research group CEA began in 2004 to develop a MATLAB KRL toolbox. This toolbox supports control programming of KUKA robots within MATLAB including the usage of all available MATLAB features. Moreover, it provides a first MATLAB based CAR system [7]. As well as the KCT, the MATLAB KRL Tbx was limited to KUKA robots, but users already benefited from the powerful methods as well as the various interfaces provided by MATLAB and its toolboxes.

Almost all of the proprietary robot languages are imperative languages containing similar programming elements. Thus, the approach of the MATLAB KRL toolbox was generalized. A uniform robot control language for robot types from different manufacturers has been developed with the RCV Tbx for MATLAB. Moreover, the simulation and 3D visualization tools have been enhanced.

This paper is organized as follows: Section 1 introduces the concept of RCP and relates it to robot controls. In Section 2, the RCV Tbx for MATLAB is described. Design, implementation as well as user interface aspects are analyzed. Finally, Section 3 gives a summary and identifies potentials for future work.

1 Rapid Control Prototyping

This section summarizes the RCP approach and how it can be used for robot control development. Systematic development of controls can be carried out following the V-model derived from Orth, Abel and Bollig [8, 3].

The V-model defines two main phases, the design phase and the commissioning phase. The design phase starts with the problem specification and continues with a draft and simulative testing of the control algorithms.

It is completed with coding of an executable control program for the target hardware. This piece of software is then used to bring the control into operation. The commissioning phase starts with component tests and ends with a test of the control across the entire process. Each step of the V-model may have to be performed several times and through several iterations or it is possible that leaps back in the development process will occur.

1.1 Fundamentals

RCP generally requires either a well-adjusted tool chain to follow the V from specification phase down to coding phase and up to operational phase or support from an integrated development environment. This integrated development environment can be an SCE. Furthermore, Software-in-the-Loop simulation (SiL) and Hardware-in-the-Loop Simulation (HiL) techniques following Abel and Bollig [3] and explicit automatic code generation are key features of RCP-capable software systems.

Maletzki [9] adopted the general V-model for controls for robot control development as illustrated in Figure 1.

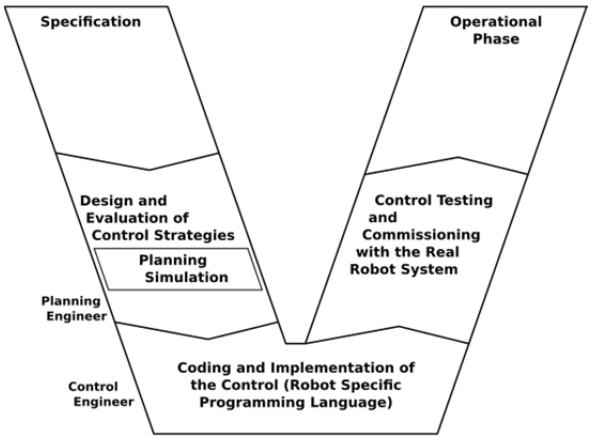

Figure 1: Adopted V-Model for Robot Control Development According to Maletzki [9].

A notably critical point is the transfer of results from planning simulation to the coding and implementation phases. As for industrial robots, the executable control program for the target hardware typically has to be written in a robot specific programming language; a continuous tool chain is not guaranteed. The control strategies that result from the planning simulation are handed over from the planning engineer to the control engineer. This kind of manual handing over is obviously fault-prone.

1.2 RCP and proprietary robot controls

Conventionally, robot controls are coded in vendor-specific programming languages and tested using dedicated 3D-simulation software. Such CAR systems offer software libraries with robot models from the particular vendor and usually include interfaces to 3D-CAD software. In CAR systems, concrete control strategies can be implemented within the specific robot programming language so that extra coding after simulative testing of the control is not required. 3D visualizations, of e.g. robot movements and potential collisions, play a decisive role if CAR systems are deployed. Continuity as required for RCP is partly given but manual knowledge transfer from planning to control engineer is still necessary.

The conventional robot control programming approach supported by CAR systems and associated languages and software respectively are depicted in Figure 2.

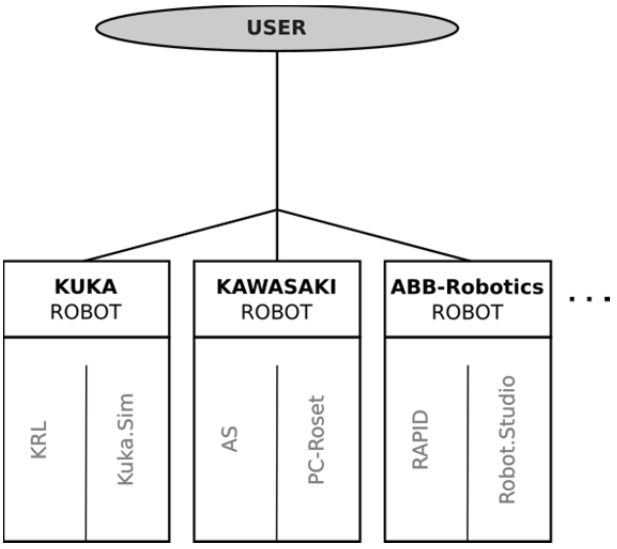

Figure 2: Conventional robot control development using vendor specific software.

After specification of the control requirements the planning engineer designs control strategies using planning simulation tools.

Subsequently, he delivers a control strategy to a control engineer mostly in a textual manner. The control engineer implements the robot control using vendor specific tools as listed in Figure 2. At this point there is a break in the tool chain according to the definition of RCP by Abel et al. [3].

However, a continuous reuse of software components during this transition is hard to realize for robot control development, because different concepts are used and are necessary in these two phases. Starting from this point, a control engineer can implement and deploy a robot control for vendor specific robots compliant with the definition of RCP. For example, for KUKA robots, coding of controls is done using KUKA Robot Language (KRL) and simulation of the control with robot models is carried out using the Kuka.Sim software. Control testing and commissioning with the real robot system can be achieved by automatic code generation or a communication link as illustrated in Figure 3.

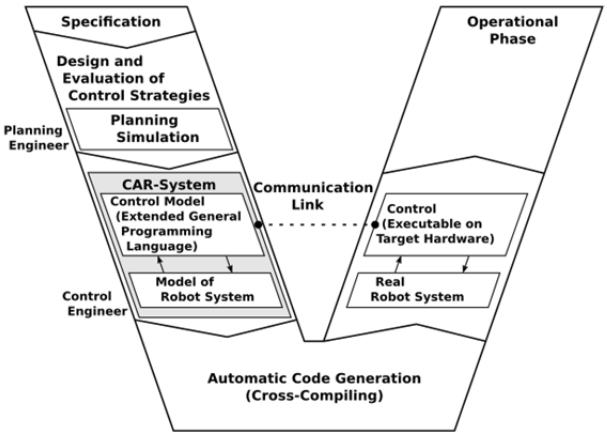

Figure 3: Detailed V-Model for robot control development according to Maletzki [9].

In the case of industrial robots, an explicit code generation for target hardware is not necessary. The control program from the design phase is implemented in a robot-oriented language and running on a robot control computer that is used in operation phase. In the context of RCP, this approach is called *implicit code generation*. Hence, commissioning can be carried out via a simple communication link between a CAR system on a PC and a robot controller. This practice is called *Software in the Loop* (SiL) approach in [3]. The concept of explicit automatic code generation for target hardware is important for mobile robotic applications.

However, the robot control development software depicted in Figure 2 is vendor-specific and incompatible between types. This fact complicates the previously discussed control development or makes it nearly impossible to develop multivendor robot applications.

1.3 RCP and the RCV Toolbox for MATLAB/Simulink

Controls for interacting robots and multivendor robot controls are examples of advanced tasks in robot control development.

Figure 4 illustrates how the RCV Tbx eases such tasks by providing not just a generalized interface to robot control and robot visualization commands, but also to robots from different vendors and/or types.

The control engineer implements a robot control as a MATLAB coded sequence of control commands. These commands are independent from a specific robot vendor as well as from a specific robot language, but they are very similar to established robot programming languages such as KRL and AS. Furthermore, control commands can be combined with any MATLAB commands and can also be generated from Stateflow or Simulink models. This option eases the implementation of complex controls by making available high level programming concepts.

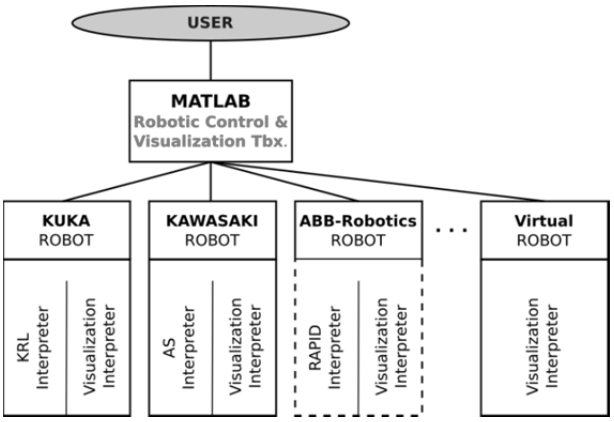

Figure 4: Robot control programming with the RCV Tbx for MATLAB/Simulink.

Moreover, the RCV Tbx for MATLAB provides robot-specific interpreters coded by using the vendor-specific robot languages, which have to be installed on the robot controllers.

The MATLAB based control PC and the robot controllers are connected via a bidirectional communication link. The interpreters are responsible for the identification and execution of control commands that are transmitted by the MATLAB based control PC and they deliver acknowledgements or sensor signals back. Figure 5 illustrates an RCV Tbx-based multi-robot configuration.

One or more computers with MATLAB/Simulink and the RCV Tbx installed act as the continuous software environment from the early design and evaluation phase via control testing with the real system and finally the operational phase. No recoding or reimplementation is necessary, rather the control program can be extended successively until it meets the de-mands of the intended control task. SiL and HiL approaches as defined in [3], as requirements for RCP capability of control development, are met because the simulated control can be stepwise extended to become the real control for the operational phase.

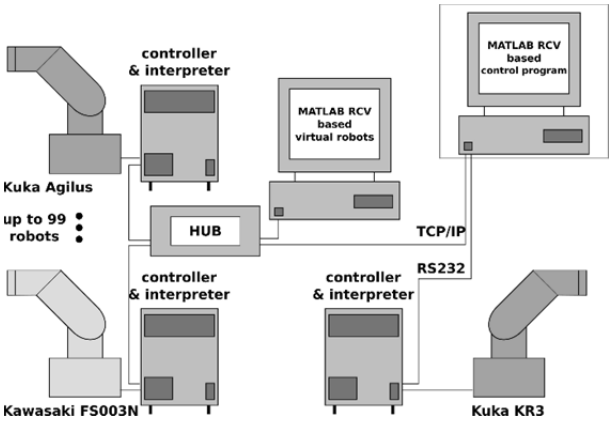

Figure 5: A Multi-robot configuration with heterogeneous robot types using the RCV Tbx.

Short control sequences or single control tasks can already be tested with the real process during design and evaluation (SiL). HiL tests, in which a real control device is tested with a simulated process, are not usually relevant for industrial robot controls as mentioned in Section 1.3. However, they could be of interest for other devices in a robot application. Furthermore, a control application can also be tested in a purely virtual environment using the RCV Tbx similar to the usage of a CAR system. For this purpose the RCV Tbx provides 3D-models of real robots and other virtual components.

2 Design and Usage of RCV Tbx for MATLAB/Simulink

This section describes and analyzes the software design of the RCV Tbx. It details the two toolbox downloads as they are available at [2], namely the robot control download (***Robotic Control Tbx***) and the visualization download (***Robotic Visualization Tbx***).

It is shown how a robot control program can be developed, tested by simulation, put into service (soft commissioning) and finally used as a real control. Continuity of the tool chain, as is needed to meet the requirements of RCP, is given. Hence, the same sequence of control commands coded in the same language can be sent either to a visualized or a real robot.

Figure 6 depicts the main functional parts of the RCV Tbx. *Control*, *Interpreter* and *Visualization* are the three main functionalities which are distributed across the two software packages.

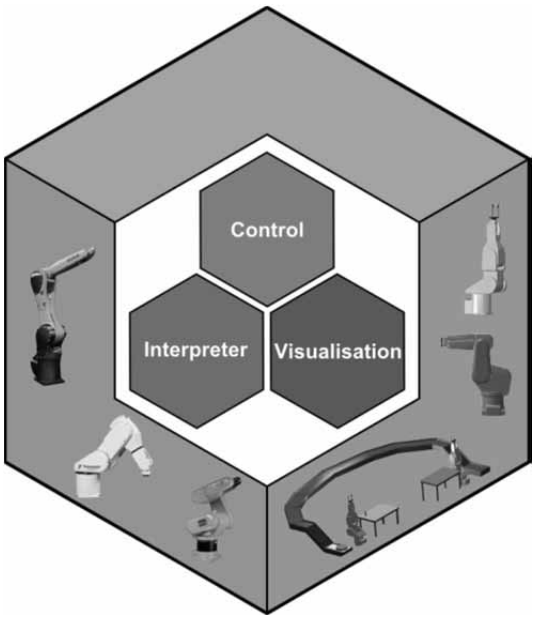

Figure 6: Main parts of the RCV toolbox for MATLAB/Simulink.

A collection of control commands for real, as well as visualized, robots and interpreters for real robots are dowloaded with the first part of the RCV Tbx, the *Robotic Control Tbx*. Interpreters for visualized robots, also called virtual robots, and the visualization software are dowloaded with the second part of the RCV Tbx, the *Robotic Visualization Tbx*. Currently, both parts support interpreters for Kawasaki FS003N, Kuka Agilus KR6 (TCP/IP connection) and for KUKA KR3 (RS232 connection).

Interaction between the MATLAB control PC and the interpreter program as well as the interpreter algorithm is depicted in Figure 7.

The interpreter can be installed either on a robot controller or on a MATLAB PC as part of the *Robotic Visualization Tbx*. Implementation of the interpreter slightly differs depending on robot type and robot controller hardware. However, the basic principle of interpreter program algorithm remains the same.

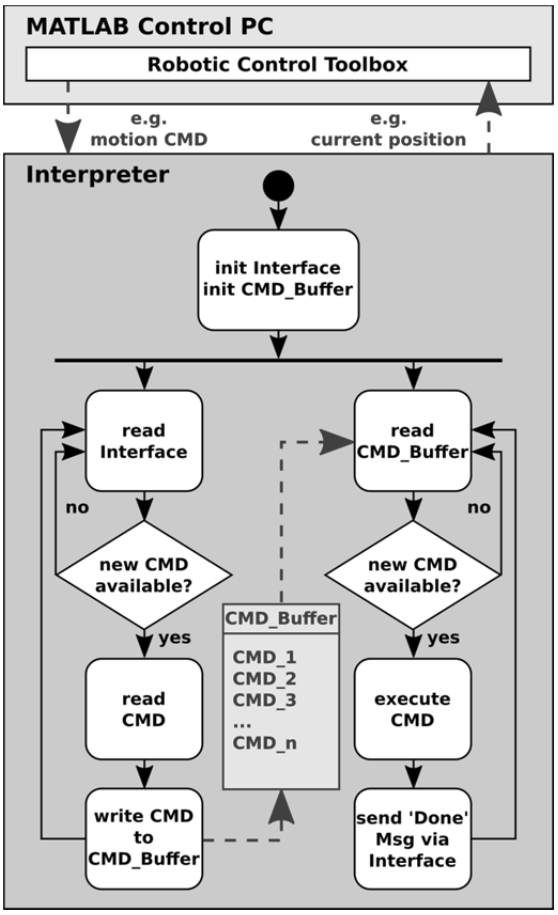

Figure 7: Simplified interpreter algorithm.

The connection between interpreter and MATLAB control PC is always established by the interpreter. It can either be an RS232 or a TCP/IP link. After connection is established and communication interface as well as the command buffer is initialized, robot control commands can be received by the interpreter. Commands are written to a FIFO buffer, read from there and brought to execution. Buffering the commands is necessary, because execution of most commands takes some time.

The interpreter program is not blocking meanwhile and receiving of subsequent commands is always possible. Some commands, such as commands for stopping the robot for security reasons or altering the motion speed during an onging robot movement, have to be executed immediately. Handling of those commands is not shown in Figure 7.

2.1 Part I: Robotic Control Tbx

This subsection details robot control programming and gives some examples of robot control commands.

The control of real robots requires an interpreter program to be installed on the robot controller. The interpreter for each kind of real robot is written in the appropriate robot-specific language and copied to the robot controller. The interpreter program takes the commands from MATLAB and translates them into commands for the robot-specific language which are executable by the robot controller. Security considerations, such as workspace supervision and movement execution, are, therefore, still covered by the robot controller.

After the toolbox is installed on the control PC and the appropriate interpreter is copied to the robot controller, the interpreter program on the controller needs to be started. For RS232 connections, after startup of interpreter, the controller is ready to receive MATLAB control commands immediately. For TCP/IP connections, the interpreter on the controller acts as a server and waits for a suitable client to connect. This client is a robot object created on the control PC.

Table 1 lists all available commands for robot control alphabetically.

Of essential importance is the `robot()` command which creates and destroys the robot object which acts as an interface for control commands. Listing 1 shows the syntax of the `robot()` command. Currently, robots can either be of type'Kawasaki' or of type 'Kuka'.

```
>> h1 = robot( 'open', 'Kawasaki',...
               'tcpip', IP-ADRESS, PORT );
>> h2 = robot( 'open', 'Kuka',...
               'serial',COM-PORT);
>> robot( 'close', h1, h2);
```

Listing 1: Create and destroy robot objects.

rbrake()	brake the motion of one or all robots directly
rcallback()	define functions, that are executed automatically
rdisp()	formatted display of position structures
rerror()	set up a function, dealing with error codes received from robot controllers
rget()	get interpreter-, toolbox- and motion-properties and current positions
ris()	check the status of commands and processes
rkill()	brake and stop the motion of one or all robots directly; then set up the original state of robot(s)
rmove()	move a robot
robot()	create or destroy a robot object that can be controlled
rpoint()	define robot-positions with coordinates and motion-properties
rprocess()	define complex operations
rreset()	reactivate robot(s) after automatic switching-off of the interpreter(s)
rrun()	reactivate robot(s) after using rbrake() or rstop()
rset()	set up interpreter-, toolbox- and motion-properties
rstatus()	switch ON or OFF status report
rstop()	brake and stop the motion of one or all robots directly
rteach()	teach and save positions of a robot
rwait()	block the MATLAB-prompt until the status of commands or processes fulfills a condition

Table 1: List of available robot control commands.

The command `robot()` creates a robot object and returns a MATLAB handle to the object. This handle can then be passed to other control commands to address this specific robot. Some control commands such as `rbrake()`, `rkill()`, `rreset()`, `rrun()` and `rstop()` affect all robots that the control PC is currently connected to, if no handle is passed to the command as a first parameter. Complex control operations can be coded by using the command `rprocess()`. With this command, tasks to be fulfilled by more than one robot can also be defined. Sequential as well as concurrent operations are possible, and control commands can be structured and grouped. Listing 2 shows a short example for a concurrent operation with two robots.

```
>> load P1 P2
>> rprocess( {...
              Kuka, P1,...
              Kawasaki,{P2, 'speed', 50},...
             },...
             {
              Kuka, 'home',...
              Kawasaki, 'home',...
             } );
```

Listing 2: Two robots in a concurrent operation pro-grammed with `rprocess()`.

`Kuka` und `Kawasaki` are the handles for the robots. The robots move at the same time to the positions `P1` and `P2` which are defined by the loaded variables `P1` and `P2`. They start to move to their home positions only after they have both finished their moves.

The `rprocess()` command already allows some complexity, but furthermore, all standard and advanced programming features of MATLAB/Simulink can be used to create even more complex and also simulation based robot controls. An example of such integration with other tools is depicted in Figure 8 where RCV control commands are embedded in Stateflow.

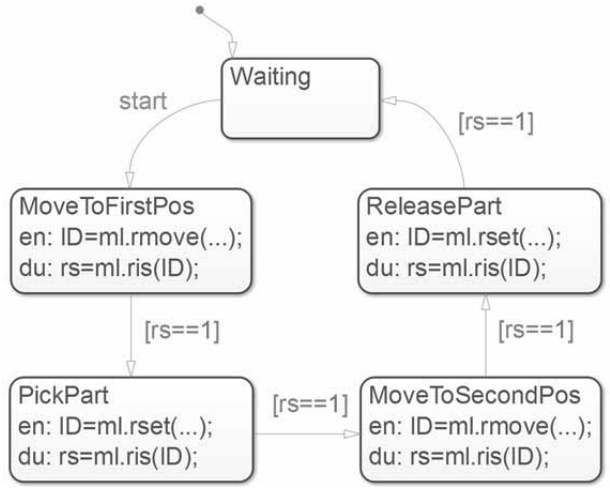

Figure 8: State-based control using Stateflow and RCV Tbx.

The integration of arbitrary external hardware, which can act as sensors or actors, is also feasible. This would allow a big advantage compared to the restricted availability of additional hardware when developing robot controls with proprietary software and languages.

2.2 Part II: Robotic Visualization Tbx

The *Robotic Visualization Tbx* is the second software package of the RCV Tbx and can be used for testing and enhancement of controls developed with the Robotic Control Tbx. It offers distributed 3D-visualization of up to 99 robots in MATLAB and allows interactive control of visualized real robots and also pure virtual robots in a virtual environment. Thereby, safe development, testing and debugging of robot applications is possible. Furthermore, the toolbox includes an STL interface for importing user-defined graphical objects designed using external CAD software.

The control of a visualized robot requires an interpreter program, too. Interpreter functionality for virtual robots is included in the *Robotic Visualization Tbx*. Interpreters are part of virtual robot objects and establish a TCP/IP link between control PC and visualization PC. Control program and visualization can physically be located on the same PC, represented by two MATLAB instances. User toolbox's interface as well as virtual objects is designed as a MATLAB class. Figure 9 pictures visualizable object types that map entities of the real world.

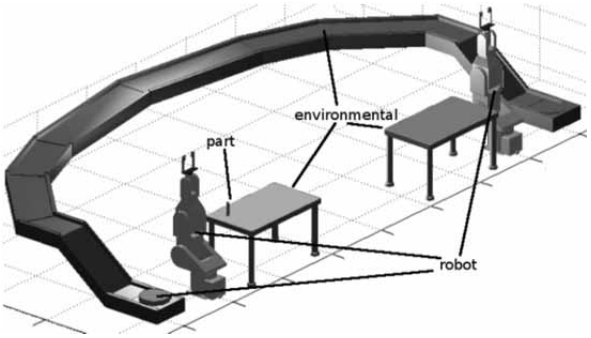

Figure 9: Types of visualizable object.

Besides virtual robot objects environmental objects and part objects can be visualized. Environmental objects are passive objects that cannot be moved by the robots. Part objects are also passive objects, but they can be picked and moved by robots. In Figure 9 examples for environmental objects are two tables and a conveyor; a test tube represents the part objects. Robot objects that represent the active visualization objects include kinematics which matches the kinematics of the corresponding real robot. Furthermore, pure virtual robots or other active objects such as carts can be visualized if the user defines appropriate kinematics. Figure 9 shows two visualized real robots and a pure virtual cart.

Table 2 lists commands of the Robotic Visualization Tbx. The first three commands are used to initialize, finish and monitor a visualization session. With the creation of a MATLAB 3D-figure at startup, a MATLAB timer object is also started. It ensures that the visualized objects are refreshed 20 times per second to achieve a smooth appearance of the animation.

ViSu.start	initialize the visualization, open an empty 3D window
ViSu.stop	stop the visualization, delete all virtual objects, close the figure
ViSu.info	display all virtual objects with their ID, type, position and additional information depending on object type
ViSu.create()	instantiate a robot object of type 'Kuka', 'Kawasaki' or user defined type
ViSu.repose_robot()	alter position of robot object identified by its id
ViSu.delete_robot()	delete a robot object identified by its id
ViSu.place_env()	instantiate an environmental object
ViSu.repose_env()	alter position of an environmental object identified by its id
ViSu.delete_env()	delete an environmental object identified by its id
ViSu.place_part()	instantiate a moveable object
ViSu.repose_part()	alter position of a part object identified by its id
ViSu.delete_part()	delete a part object identified by its id

Table 2: User Interface for Visualization.

The other commands offer identical functionalities for the three different types of visualizable objects: robot, environmental and part objects can be I) initialized, II) repositioned, and II) deleted during a simulation session.

Currently, the toolbox is being revised extensively to harmonize the user interface and improve software stability and robustness.

2.3 Integrated RCV Tbx usage example

Figure 5 introduced in Section 1 shows an example of multi-robot configuration. Notice that in that configuration we have integrated real robots from different vendors, i.e. from Kuka (Agilus, KR3) and Kawasaki (FS003N) and some virtual robots, too.

In this section we focus on an academic scenario where we have a robot control being applied only to some virtual objects. For this, it is necessary to start two instances of MATLAB; of these, one acts as the control PC (client) and the other is the visualization server. These MATLAB instances can either be located on different computers, as shown in Figure 5, or on the same computer as in the following example. The example includes two robots, the environmental object table and a test tube which is is classified as a 'part'.

Listing 3 illustrates how MATLAB control and visualization instances can interact. MATLAB commands in line 1, 2, 4, 6, 7, 11 to 12, and 14 have to be executed on the visualization instance, while the indented lines 3, 5, 8 to 10, and 13 are control commands which have to be executed on the control instance.

```
1  >> ViSu.start;
2  >> ViSu.create('Kawasaki', 40000,...
                  [0,0,0,0,0,0]);
3      >> r1=robot('open', 'Kawasaki',...
              'tcpip', 'localhost', 40000);
4  >> ViSu.create('Kuka', 40001,...
                  [500,500,0,0,0,0]);
5      >> r2=robot('open', 'Kuka',...
             'tcpip', 'localhost', 40001);
6  >> ViSu.place_env('table.stl',...
                   [-800 0 0 0 0 0],'blue');
7  >> ViSu.place_part('test_tube.stl',...
               [-800 0 400 45 0 0],'white');
8      >> rset(r1,'signal', [-9, 10]);
9      >> rmove(r1,'home2');
10     >> rmove(r2,'home2');
11 >> ViSu.info;
12 >> ViSu.delete_part(1);
13     >> robot(r1, r2, 'close');
14 >> ViSu.stop;
```

Listing 3: Code Example of Interaction of the two Parts of RCV Tbx for MATLAB/Simulink.

After a virtual robot has been created with the command `ViSu.create()`, the MATLAB prompt for the visualization instance is blocked until the TCP/IP connection to the control instance is established. This is accomplished when the appropriate `robot('open',…)` command is executed on the control instance which initiates a robot control object. The table, as well as the test tube, just exist virtually and are passive objects that are not controllable and therefore have no counterpart on control instance. After line 7 is executed, the visualization looks as depicted in Figure 10.

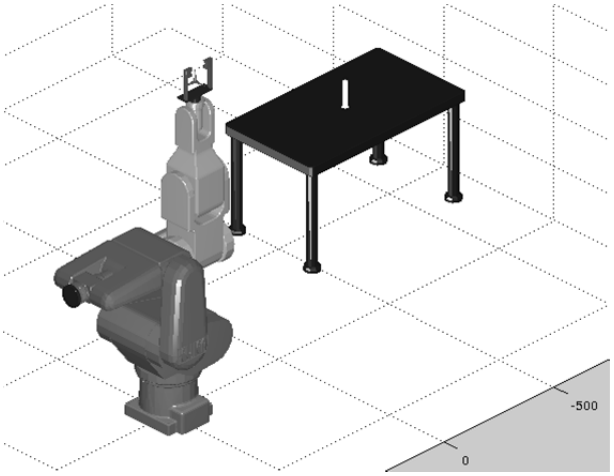

Figure 10: Visualization example after creation of objects.

Then some simple control commands, beginning with line 8, are executed. The command in line 8 closes the gripper of the Kawasaki robot. After that, both robots are moved to one of their predefined home positions, `'home2'`. Back on the visualization instance, information on the current visualized objects is requested. With this information the user knows, that the test tube is a 'part' object with ID 1 and can be deleted during the visualization session. This feature is useful, if one wants, for example, to alter the surroundings of active robot objects without restarting the visualization.

After execution of line 12 the scenario looks as depicted in Figure 11. The gripper is closed, both robots have moved to their home positions and the test tube has disappeared. The last two commands close the TCP/IP connections and finally close and delete the visualization figure.

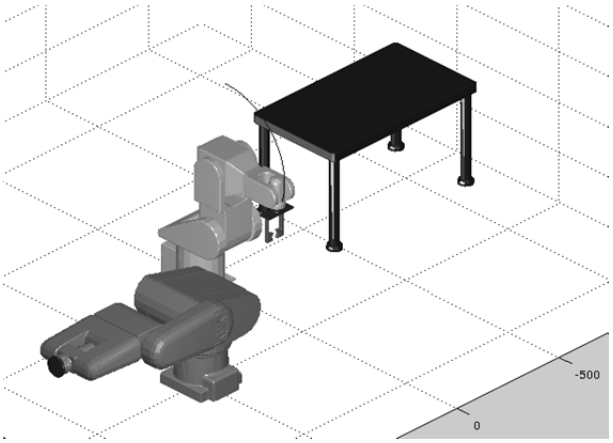

Figure 11: Visualization example after execution of some control commands.

Without larger modifications it is possible to apply the same sequence of control commands to real robots in a real environment, if we assume the real robots are of appropriate types and placed in the same positions. In this case, instead of the commands `ViSu.create()`, that initialize the visualized real robots, TCP/IP and RS232 connections to the control PC need to be established by the robot controllers. For visualized robots all connections are of type TCP/IP, although for a real KUKA KR3, for example, an RS232 connection is necessary. Hence, the control command in line 5 has to be adapted. The control object `r2` needs to be opened with the parameter `'serial'` instead of with `'tcpip'`. Instead of a visualized table and a visualized test tube, a real table and a real test tube could be placed in the real robot cell. Motivation for such simple control tests by simulating the movements first in a virtual environment could be, for example, to avoid collisions between robots and table.

3 Summary and Further Work

The RCV Tbx offers excellent possibilities for developing multivendor robot controls in a homogeneous software environment. It fulfills the main requirements of a RCP capable tool. The key benefits of using the RCV Tbx are the possibility to develop monolithic control programs for interactive robots from different manufacturers and the easy, manufacturer-independent integration of external hardware.

The MATLAB/Simulink environment is well established in the area of engineering and, today, is a standard tool for engineers. Hence, the engineer can benefit from employment of all programming tools available in this environment. Especially for recent advanced control applications such as the Simulation Based Control (SBC) approach introduced by Maletzki [9] the integration of the RCV Tbx into MATLAB/Simulink is very advantageous. Simulation models are directly used as control programs in this approach and it is obvious that RCV Tbx control commands can easily be integrated into these kinds of simulative control.

If one takes another step, it can be seen that models employed for control tasks can also be automatically generated from a knowledge base as proposed in the SBC and System Entity Structure (SES) approach by Schwatinski and Freymann [10, 11]. The development of flexible, task oriented multi-robot controls is considerably simplified with this approach.

References

[1] Pawletta T, Freymann B, Deatcu C, Schmidt A. Robotic Control and Visualization Toolbox for MATLAB. In: Breitenecker F, Kugi A, Troch I, editors. *MATHMOD 2015.* Proceedings of MATHMOD 2015 - 8th Vienna Int. Conf. on Mathematical Modelling; 2015 Feb 18.-20., ARGESIM Report No. 44, ARGESIM, Vienna/Austria UT; 2015. P. 371-372 & Poster.

[2] Research Group CEA. (2011). *Robotic Control & Visualization (RCV) Toolbox for MATLAB* [Internet]. [cited 2015 March 28]. Available from: http://www.mb.hs-wismar.de/cea/sw_projects.html

[3] Abel D, Bolling A. *Rapid Control Prototyping – Methoden und Anwendungen.* Springer-Verlag Berlin. 2006.

[4] Visual Components Corporation (2014). *3DRealize-R* [Internet]. [cited 2014 Oct 24]. Available from: http://www.visualcomponents.com/products/3drealizer-r

[5] Chinello F, Scheggi S, Morbidi F, Prattichizzo D. KCT: a MATLAB toolbox for motion control of KUKA robot manipulators. In: *Proceedings of IEEE Int. Conf. on Robotics and Automation*, Anchorage, Alaska, 2010; P. 4603-4608.

[6] Chinello F, Scheggi S, Morbidi F, Prattichizzo D. KUKA Control Toolbox. Motion control of robot manipulators with MATLAB. *Robotics Automation Magazine*, IEEE, 2011; 18(4):69-79.

[7] Maletzki G, Pawletta T, Pawletta S, Lampe BP. A model-based robot programming approach in the MATLAB/Simulink environment. In: *Advances in Manufacturing Technology – XX*, 4th Int. Conf. on Manufacturing Research (ICMR06); 2006 Sept. 05-07; Liverpool, UK, 2006. P. 377-382.

[8] Orth P, Bollig A, Abel D. Rapid Control Prototyping diskreter Steuerungen in der Automatisierungstechnik. In: *SPS/IPC/Drives Congress*; Nürnberg, Germany; 2004. P. 143-152.

[9] Maletzki G. *Rapid Control Prototyping komplexer und fexibler Robotersteuerungen auf Basis des SBC-Ansatzes.* [dissertation in German]. Rostock University, Germany, 2014.

[10] Schwatinski T, Pawletta T, Pawletta S. Flexible Task Oriented Robot Controls Using the System Entity Structure and Model Base Approach. In: *Simulation Notes Europe (SNE)*, 2012; 22(2): 107-114.

[11] Freymann B, Pawletta T, Schwatinski T, Pawletta S. Modellbibliothek für die Interaktion von Robotern in der MATLAB/DEVS-Umgebung auf Basis des SBC-Frameworks. In: *Proceedings of ASIM-Treffen STS/GMMS*, Reutlingen 20./21.02.2014 - ARGESIM Report Nr. 42, ASIM Mitteilung AM 149, ARGESIM/ASIM Pub. Vienna, Austria, 2014, P. 199-208.

Acknowledgement

The authors are thankful to our former co-worker Tobias Schwatinski and all students who worked very motivated within the project. Furthermore, the authors acknowledge the financial support of the Federal Ministry of Education and Research and the Ministry of Education, Science and Culture of Mecklenburg-West Pomerania.