Trajectory Description Conception for Industrial Robots.

Sergey Alatartsev
Computer Systems in Engineering, Otto-von-Guericke University of Magdeburg, Germany

Matthias Güdemann
INRIA Rhône-Alpes, France

Frank Ortmeier
Computer Systems in Engineering, Otto-von-Guericke University of Magdeburg, Germany

Abstract

In this paper we observe the difficulties one can face, while wiling to obtain a complicated robot movement, like using multiple standalone or implemented in different MPLs (Motion Planning Library) algorithms. We propose a new conception and a language whose goal is to solve these problems. The idea is to present an interface between robot programming instruments and existing algorithms. In contrast to the existing related methods, we propose approach based on the declarative language (without control flow) for a trajectory specification. Our goal is to provide a powerful tool for developers of software approaches for industrial robots programming. It should allow them to obtain difficult motions by easy combination of different MPLs in one application using unified specification of the movement. In addition, the proposed conception hides the inner structure of libraries and eliminates the need to investigate algorithms before applying. That would increase the speed and the quality of the newly developed software systems.

1 Introduction

The increasing rate of production and the shortening life cycle of the products are the trends of modern manufacturing process. At the present moment industrial robots do not meet these demands. Although industrial robots are multipurpose and are able to do numerous kinds of work, they are usually only used to do a single task for a very long period of time. As far as a change of this task would cause a lot of additional programming expenses. After analyzing different programming methods for industrial robots for the last 20 years, we have come to a conclusion that modern approaches in computer programming are not actually applied in industrial programming. This point of view is also shared by [24][23][22].

This paper is focused on the difficulties of motion planning realization as we believe that it is one of the main obstacles on the way of developing new approaches for industrial programming. We expect to eliminate these difficulties by introducing a Trajectory Description Conception (TDC). We mostly aim our research at the utility of TDC for developers of software for programming industrial robots rather than at programmers who directly operate the machines. We also provide a tool - Trajectory Description Language (TDL) to implement TDC ideas.

There are a lot of generic simulation environments (e.g. Workspace [26], Cosimir [21]) and robot manufactures software (e.g. ABB RobotStudio [20], KUKA-Sim [19]) that have path planning features and some of them could optimize robot movements. Most of the products implements existing algorithms by their own. Even large companies experience difficulties in realization of motion planning functionality, and following [8] there is no currently available offline environment that could provide a full motion planning functionality. While robot manufacturers and large commercial software development companies can afford to create their own Motion Planning Libraries (MPL), for small companies or researchers it may become a bigger issue. Introducing path planning functionality is a large barrier while developing new programming approaches. This problem is well known and within last years a lot of motion libraries appeared. MPK (Motion Planning Kit) is a C++ library that was developed in Stanford university and allows to use arbitrary number of robot and obstacles in path planning. Further we will refer to it as MPKit in oder not to mix it with another library MPK (Motion Planning Kernel)[18]. In addition to conventional planners, MPKernel also have the planners for the general end-effector constraints. MSL (Motion Strategy Library) [17] was developed in University of Illinois and provide planners like: Rapidly-exploring Random Trees, Probabilistic Roadmaps, and forward dynamic programming. KineoWorks [16] is a software that provides features for path planning, collision detection, path optimization. It is not available as open source. ROS (Robot Operating System) [15] is a framework for robot software develop-

ment. It includes many different motion planning components: CHOMP [14], OMPL [13], SBPL [12], STOMP [11]. CHOMP (Covariant Hamiltonian Optimization and Motion Planning) plans very smooth trajectories that lay away from obstacles. OMPL (The Open Motion Planning Library) that was developed in Rice University is a nutshell that includes a set of path planning algorithms with the possibility for extension. SBPL (Search-based Planning Library) consists of motion planners that are built around search based planning algorithms. These planners can generate a path from any current point to a target by combining several primitive motions. STOMP (Stochastic Trajectory Optimization for Motion Planning) allows a generation of collision-free and smooth trajectories that could be optimized for arbitrary secondary criteria like energy or time. It allows the usage of constraints such as holding the gripper horizontally for a given time.

Despite the fact that there exists a number of open source libraries (except KineoWorks software), their usage causes another sort of problems. Nowadays, using one MPL can be not enough while developing a new approach for industrial robot programming as far as very often various tasks force to involve multiple algorithms for solving. Developer has to consider several different types of input parameters and adopt his program for several different output types as well. In addition, developer should know what algorithms are currently needed to perform a specific motion and their right sequence of application. The difference of inner structure of MPLs causes difficulties with interchangeability. Solution to make libraries interchangeable and allow benchmarking was proposed by [10]. It is based on the idea of refactoring technique for existing libraries in order to restructure them. It could be done by harmonizing inner architecture in such a way that it does not change their external behavior. Proposed guidelines certainly could help to overcome these problems and allow a benchmarking of different algorithms. However, when the benchmarking is not needed, refactoring of existing architectures becomes unnecessary and tremendous work. For that case we offer TDC that could allow having a unique interface for accessing different MPLs without refactoring and obtaining complicated trajectory without having a deep knowledge of path planning algorithms.

Detailed information about existing robot programming approaches could be found in these surveys [9, 8].

The remainder of the paper is organized as follows. TDC and TDL are discussed in the section 2. Section 3 provide an example that shows the benefits of using TDC. In the section 4 we reason about offered conception and future work.

2 Description of TDC

We propose Trajectory Description Conception that consists of two components:

- Trajectory Description Language (TDL) and its solver. TDL is a declarative language, that would allow an elimination of control flow. It consists of a set of constraints that are used to describe the motion.
- Facade API uses ideas of a design patterns method [7] that is already proved and widely used in computer software engineering. It is based on principles of the Facade Design Pattern. This pattern allows wrapping a poorly-designed APIs into one well-designed API. It also reduces dependencies of outside code and inner code of a library.

TDC is an intermediate layer between algorithms (standalone or in MPLs) and existing industrial robot programming approaches. Currently in case of using different path planning libraries, industrial robot approaches have to know specific input and output interfaces as well as what exact algorithms should be applied. Core task of TDC is to provide an input and an output interfaces in the most generic way, so that it would be possible to use TDC with major existing industrial robot programming approaches. In addition, TDC hides the inner structure and details of realizations of MPLs. The principal difference between TDC and existing approaches is that in TDC input parameters are specified as constraints. This specification should not depend either on task (e.g. in contrast to [25], where programmer concentrates efforts on the application-specific problem) or underlying algorithms.

The planning process could be divided into the five stages (follow **Figure 1**): 1) High-level program or a programmer specifies with TDL the preferences of how trajectory should look like; 2) Solver observes arbitrary input set of constraints, decomposes it into a subtasks, then requests the facade for a certain algorithms that could resolve them; 3) Facade query different algorithms to solve the problem; 4) Underlaying algorithms from different MPLs produce the solutions; 5) Different parts of the solution are composed into one trajectory, that is returned to the requester.

We strongly believe that deep knowledge of path planning algorithms and especially inner structure of their realization is not necessary for industrial robot software programmer and it only brakes process of development. In general, while using TDC there will be no need for developer to know inner structure of path planning libraries and to know what algorithms are needed in a certain situation and how to combine them in order to get a complex trajectory.

2.1 Trajectory Description Language

Before talking about the TDL, several preliminaries should be given.

A path τ is a function that maps a parameter from $[0, 1]$ to a joint space Q, i.e. $\tau : [0, 1] \to Q$. Conventionally industrial robots have six degrees of freedom,

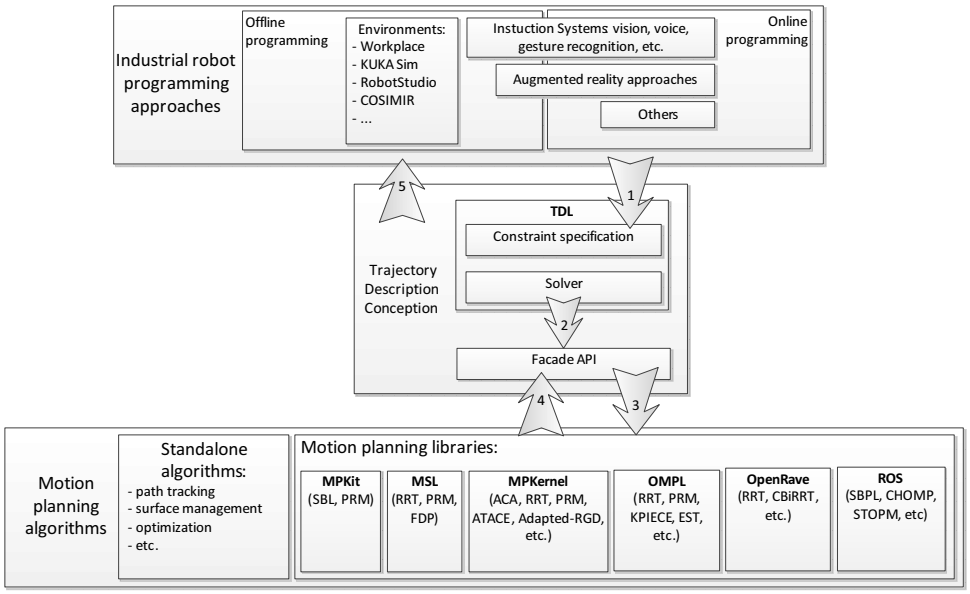

Figure 1: Trajectory Description Conception

i.e. $Q = R^6$. Motion law $s(t)$ is a time scaling function. It maps time $\forall t \in [t_0, t_{max}]$ to the path parameter: $s : [t_0, t_{max}] \rightarrow [0, 1]$. A composition of the motion law and the path produces trajectory: $\tau(s(t))$.
It is not easy to provide a very generic input and output that would be suitable for all high-level applications as well as will be able to describe most of the tasks performed by industrial robots. For that purpose we offer a new approach - Trajectory Description Language. The basic idea of the TDL is to get a trajectory $\tau(s(t))$ in joint space (output) after applying constraints for a movement (input) that eliminates undesired behavior. The key challenge while creating TDL is to identify classes of the constraints in a such a way that following requirements would be fulfilled:

- A set of described constraints should be enough to describe major of the common motions.
- The constraints should be organized in such a way that this specification could be applied in major MPLs.

With TDL a programmer is not interested what particular movement robot will execute to perform a task. TDL differs with currently used languages like: Epson Robotics SPEL+ [6], Stäubli VAL3 [5], ABB Rapid [4] which are low level and imperative. It is intended that TDL will eliminate all control flow and will have possibilities to optimize trajectory, in contrast to other high-level languages like AUTOPASS [3] and RAPT [2]. By basing on existing MPLs, TDL will allow constructing of collision free trajectories that RAPT cannot provide.

2.1.1 Constraints specification

Temporal Constraints. Atomic time constraint is a certain point t in time. It is supposed that program starts working at time t_0 and stops at t_{max}. Temporal constraint could be used in combination with other constraints, e.g. for limiting their active working time. Let's call interval of time $[t_1, t_2]$ (where $t_1 \geq t_2$) when the constraint is active - *Active window*.
Velocity constraints. In TDL programmer expresses only his expectations of how velocity should look like. For that it is possible to define three different types of velocity constraints: velocity $V(t)$, acceleration $V'(t)$ and jerk $V''(t)$ constraints. After applying these constraints TDL will find the most appropriate velocity function, in case if it exists.
Spatial Constraints. Position of a TCP (Tool Center Point) is a 6D tuple (x, y, z, n, s, a) where (x, y, z) describes a Cartesian position of the point in space and n, s, a are vectors, that show the direction of axis orientation and $n = \{n_x, n_y, n_z\}$, $s = \{s_x, s_y, s_z\}$, $a = \{a_x, a_y, a_z\}$. Based on this definition spatial constraints could be used for describing following cases:

- Constraints for (x, y, z) that normally specify forbidden areas.
- Constraints for n, s, a that eliminate unwanted rotation.
- Specific point or area to reach (see section 3).
- Definition of the type of needed motion (e.g. linear).

Optimality Constraints. TDL also includes optimization constraints, which will be used to select one of the admissible trajectories, e.g. the "best" one wrt. an optimization goal. There are three objective functions that are commonly used for optimization: time, jerk and energy consumption.
Our goal is to make a classification of the constraints as simple as possible in order to get atomic constraints. Later

atomic constraints could be combined, for example, to assign target point for path planners (spatial - to define a point and temporal - to define a time). Another possibility is to combine atomic constraints and get completely new movements. It is also a great advantage in comparison with existing approaches.
It is very important to mention that this specification is very rough and only the most widely used constraints were presented. Further research direction is to define the Minimum Sufficient Constraint Set (MSCS) that is a subset of all possible constraints. It means that every other constraint, that is not in MSCS could be represented as a combination of already defined constraints from MSCS. That could be challenging due to the numerous types of tasks and motions that should be covered with constraints, involved several spaces, and areas of influences: either for the whole robot body or end-effector.

2.1.2 TDL solver

We assume to use the decoupled approach for planning, that is widely used nowadays [1]. The idea is that motion planning process is decoupled into several stages: the first stage is to find a path τ with a certain number of applied constraints and then find a motion law $s(t)$. This approach simplify the process of planning and reduce computation problems.
TDL solver should take every arbitrary combination of constraints from the constraint space as an input and produce a trajectory.
Let's define a space of all possible constraints CS and a space of all possible trajectories TS. Then a solver could be defined as a function that maps a certain set of constraints into a trajectory: $Solver : CS \rightarrow TS$. The semantic of the solver could be defined as follows:

function SOLVER($Constraints$)
  Declare $Solution[]$ as TS
  Declare $StageCS[]$ as CS
  $StagesSC[\ ] \leftarrow$ DetermineStages($Constraints$)
  for all $StagesSC[]$ **do**
    $PathC$ = $\{pc | pc \in StageCS[i], pc \in PCS\}$
    MLC = $\{tc | tc \in StageCS[i], tc \in TCS\}$
    $\tau_i \leftarrow$ SolveP($PathC$)
    $s(t)_i \leftarrow$ SolveML(MLC)
    $Solution[i] \leftarrow \tau_i(s(t)_i)$
  end for
  $\tau(s(t)) \leftarrow$Compose($Solution[]$);
  return $\tau(s(t))$
end function

$Solution[]$ is an array, defined in TS. $StageCS[]$ is an array of sets of constraints. Function DetermineStages() - divides the set $Constraints$ into sets based on the time points. More detailed description of the decomposition process as well as algorithms proposed for usage are presented in the section 3. Every set of constraints $StageCS[]$ is divided into two constraint sets: that influence either path ($PathC$) or motion law (MLC). PCS is a space of all path constraints and TCS is a space of all motion law constraints. For both these sets the appropriate planners from Facade API are applied by using the functions SolveP() and SolveML(). In this way, solution $Solution[i]$ for each stage is obtained. Function Compose() combines solutions from the different stages, and final trajectory is returned.

3 Example of using TDL

Let's describe a simple sketch example that shows the benefits of TDC and demonstrates the way how TDL solver might work. The task is to build such a trajectory for the robot to follow with a cup of water in its gripper without spilling any of it from one side of the table (point A) to another (point B) and will hit neither the obstacle (standing between A and B) nor the table (see **Figure 2**). Further, three compulsory steps and some additional constraints will be observed. These example presents the logic of constraints specification and shows how TDL solver could work with a certain constraint set.

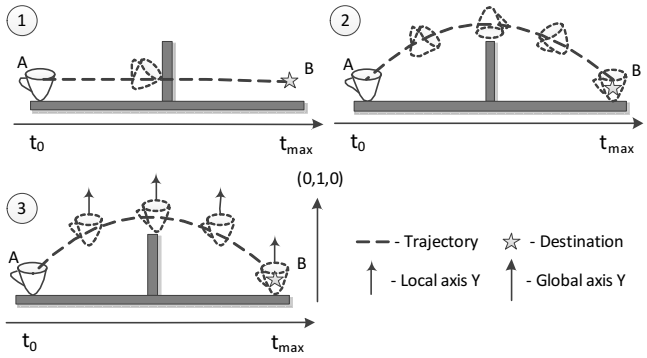

Figure 2: Applying TDC for the grasping task

The first step is to define target points by a combination of spatial and temporal constrains:
$C_1 : Spatial(A) \cup Temporal(t_1)$
$C_2 : Spatial(B) \cup Temporal(t_2)$
If only these two constraints are applied, and robot has no other information about its environment, solver will try to compute such a trajectory, that at time t_1 end-effector will visit the point A and at t_2 it will reach the point B. After receiving only C_1 and C_2 constraints, TDL solver will try to apply the most simple available algorithms, like "point to point" movement (that kind of motion is commonly used in conventional industrial robot languages) and trapezoidal velocity profile. With a very high probability the robot's tool or even the robot manipulator itself will hit other objects. Therefore on the second step, spatial positional constraint is used to specify the avoidance of obstacles:
$C_3 : Spatial(Obstacle\_geometry) \cup Temporal([t_0, t_{max}])$.
Temporal constraint is added to specify *Active window* for spatial constraint. In that case TDL solver has to find a collision free path, and could rely on probabilistic algorithms,

like RRT or PRM, that already proved to be highly efficient for high dimension spaces.
The third step is necessary, because although the trajectory is collision free and the cup reaches its destination, but due to rotation of the gripper, water would be spilled. Undesired rotation is eliminated by applying a spatial orientation constraint for the end-effector path:
$C_4 : Orientation = (0, 1, 0) \cup Temporal([t_1, t_2])$.
It forces to keep orientation (0,1,0) along the whole path. This situation allows rotating of the gripper along its Y-axis but this is not a problem, as this rotation will not cause water spilling. To solve this problem, usage of only conventional planners is not enough. Instead, usage of either ATACE algorithm from MPKernel, or CBiRRT (algorithm from CoMPS framework, that is available as OpenRave plugin) or STOMP, could solve this problem. These algorithms allow to specify the constraints on the orientation of the gripper.
Actually these three steps are sufficient to describe this motion. In addition, one could add some optional constraints, like constraints for velocity profile, that the absolute value of the acceleration should not be higher than some real number r. As a sharp jolt of the gripper could cause waves and spilling of water. This is eliminated by applying velocity constraint:
$C_5 : (TCP.Acceleration <r) \cup Temporal([t_1, t_2])$.

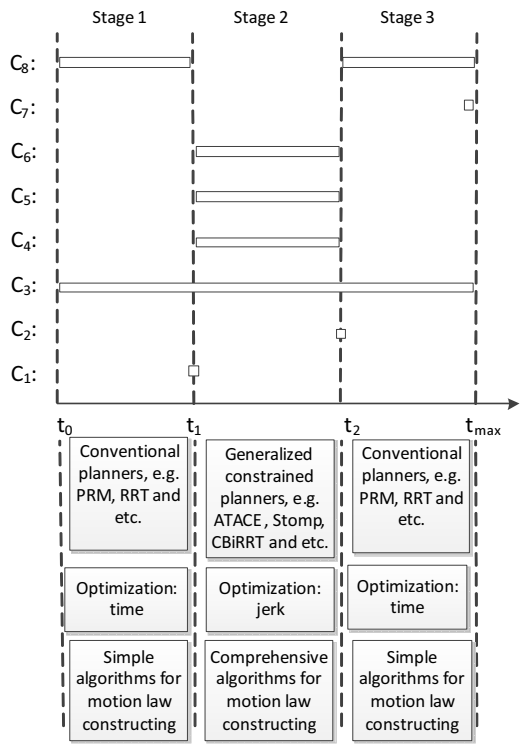

Figure 3: Decomposition of the constraints into the stages

Nevertheless, path could be not optimal from the point of time, energy consumption or jerk. Therefore, one of the optimality constraints is applied, e.g. a constraint for minimizing the jerk value in order to make movement smoother:
$C_6 : Optimization(Jerk) \cup Temporal([t_1, t_2])$
One could also want to move robot to the starting configuration when the task is done by the constraint:
$C_7 : Spatial(Start\_Configuration) \cup Temporal(t_{max})$
In case if task should be made faster, one could apply time optimization at the moments, when there is no cup in the gripper:
$C_8 : Optimization(Jerk) \cup Temporal([t_0, t_1)) \cup Temporal((t_2, t_{max}])$
Finally task could be performed satisfactorily only with first four constraints. But already with eight constraints programmer could obtain a very complicated motion, which involves several algorithms from different libraries.
To solve these constraints, TDL solver should make a decomposition of the task into several stages, based on the used time points (see **Figure 3**). Every stage indicates what constraints are currently active. For every stage set of constraints a specific set of solvers from Facade API is applied.
Proposed idea is very flexible, because if new obstacles are added, either constraint C_3 or new defined spatial constraint (i.e. C_9) should be complemented by them. If the task should be performed quicker, then the optimization parameter should be switched to *time* in constraint C_6.
The possibility to combine atomic constraints and get completely new movements is a great advantage in comparison with existing approaches. As far as now, in case of changing a task even slightly, the programmer had to rewrite the source code and define new "teach-in" points. For example, an additional object will likely make an existing trajectory invalid; TDL solver would simply calculate a new trajectory adhering to the existing spatial constraints with new spatial constraint. A traditional program based on teach-in points would require a completely new definition of points and even then additional aspects (e.g. efficiency) would not yet have been considered.

4 Conclusion

The problems of using modern MPLs and standalone algorithms were observed. A conception based on principles of a Facade Design Pattern that helps to solve them was presented. It hides the inner structure of the MPL and prevents developer of spending time to learn how to combine the algorithms and what MPLs to use. Conception provides a new language that is based on the constraint description. Presented constraints are the generalization of the most widely used nowadays. The next step is determining of Minimum Sufficient Constraint Set, that could be able to describe major of existing tasks and be self-consistent.
Applying of proposed method would provide additional functionality for offline environments, instructive systems and could give a new impulse for developing of task-oriented languages.

References

[1] LaValle, S.M.: *Planning algorithms*, Cambridge University Press, 2006

[2] Popplestone, R.J., Ambler, A.P., Bellos I.: *RAPT: A language for describing assemblies*, Industrial Robot: An International Journal, 1978, pp. 131 - 137

[3] Lieberman L.I.,Wesley M.A.: *AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly*, IBM Journal of Research and Development, 1977, pp. 321 - 333

[4] ABB: *RAPID Reference Manual*, online: http://www.abb.com/, 2012

[5] Stäubli: *VAL3*, online: www.staubli.com, 2012

[6] Epson Robots: *SPEL+*, online: http://www.robots.epson.com/, 2012

[7] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

[8] Pan, Z., Polden, J.; Larkin, N.; Duin, S.; Norrish, J.: *Recent Progress on Programming Methods for Industrial Robots*, Proceedings for the joint conference of ISR (41st Internationel Symposium on Robotics) und ROBOTIK (6th German Conference on Robotics), 2010

[9] Biggs G.; MacDonald B.: *A Survey of Robot Programming Systems*, Proceedings of the Australasian Conference on Robotics and Automation, 2003

[10] Brugali, D.; Nowak, W.; Gherardi, L.; Zakharov, A.; Prassler, E.: *Component-based Refactoring of Motion Planning Libraries*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010

[11] Kalakrishnan, M.: *STOMP (Stochastic Trajectory Optimization for Motion Planning)*, online: http://www.ros.org/wiki/stomp_motion_planner, 2012

[12] Likhachev,M.: *Search-based Planning Library (SBPL)*, online: http://www.cs.cmu.edu/∼maxim/software.html, 2012

[13] Kavraki Lab: *The Open Motion Planning Library (OMPL)*, online: http://ompl.kavrakilab.org, 2012

[14] Ratliff, N.; Zucker, M.; Bagnell, J.A.; Srinivas, S.: *CHOMP: Gradient Optimization Techniques for Efficient Motion Planning*, Proceedings of the IEEE International Conference on Robotics and Automation, 2009

[15] Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Ng, A.: *ROS: an open-source Robot Operating System.*, Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA), 2009

[16] Kineo Computer Aided Motion: *KineoWorks*, online: http://www.kineocam.com, 2012

[17] LaValle, S.; Cheng, P.; Kuffner, J.; Lindemann, S.; Manohar, A.; Tovar, B.; Yang, L.; Yershova, A.: *MSL - Motion strategy library*, online: http://msl.cs.uiuc.edu/msl/, 2012

[18] Gipson, I.; Gupta, K.; Greenspan, M.: *MPK: An Open Extensible Motion Planning Kernel*, Journal of Robotic Systems. Vol. 18, No. 8, 2001, pp. 433 - 443

[19] KUKA Roboter GmbH: *KUKA-Sim*, online: http://www.kuka-robotics.com, 2012

[20] ABB: *RobotStudio*, online: http://www.abb.com/roboticssoftware, 2012

[21] Festo Didactic: *COSIMIR® Professional*, online: http://www.festo-didactic.com, 2012

[22] Arnold, G.V.: *Automatization of the Code Generation for Different Industrial Robots*, Universidade do Minho, 2007

[23] Angerer, A.; Hoffmann, A.; Ortmeier, F.; Vistein, M.; Reif, W.: *Object-centric programming: A new modeling paradigm for robotic applications*, IEEE International Conference on Automation and Logistics, 2009

[24] Wahl, F.M.; Thomas, U.: *Introduction Robot Programming- From Simple Moves to Complex Robot Tasks*, Proceedings of the First International Colloquium "Collaborative Research Centre 562 - Robotic Systems for Modelling and Assembly", 2002, pp. 245 - 259

[25] Angerer, A.; Hoffmann, A.; Schierl, A.; Vistein, M.; Reif, W.: *The Robotics API: An object-oriented framework for modeling industrial robotics applications.*, Proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2010, pp. 4036-4041

[26] Flow Software Technologies: *Workspace*, online: http://www.workspace5.com, 2012