On the Integration of Hardware-Abstracted Robot Skills for use in Industrial Scenarios

Mikkel Rath Pedersen[1], Lazaros Nalpantidis[1], Aaron Bobick[2] and Volker Krüger[1]

Abstract— **In this paper, we present a method for programming robust, reusable and hardware-abstracted robot skills. The goal of this work is to supply mobile robot manipulators with a library of skills that incorporate both sensing and action, which permit robot novices to easily reprogram the robots to perform new tasks, as well as provide a more clear mapping between human and robot actions. Critical to the success of this approach is the notion of hardware abstraction, that separates the skill level from the primitive level on specific systems. Leveraging a previously proposed architecture, we construct two complex skills by instantiating the necessary skill primitives on two very different mobile manipulators. The skills are parameterized by task level variables, such as object labels and environment locations, making re-tasking the skills by operators feasible.**

I. Introduction

Manufacturing companies are currently experiencing a paradigm shift from mass production to mass customization; therefore, current production equipment and entire production lines need to be adjusted to follow this development, including industrial robots and automation equipment. For highly customized products, or product portfolios with large variations, a lot of manual labor is usually present. Human labor is the pinnacle of versatility when it comes to manufacturing, as a skilled worker can be shown new tasks and in most cases immediately reproduce them. In other words, if a human has a basic set of *skills*, then the repertoire of *tasks* the he/she can perform is impressively large. It is for this reason that we propose a similar structure for the mobile industrial manipulators of the future, as it is necessary to have methods for effortless reprogramming of robots at the *task level*, rather than the *robot level*, as well as provide an easier mapping between human and robot action. In task level programming, a robot program is a sequence of robot *skills* that specify a production-related goal.

To establish a fitting representation of robot skills for task-level programming has been the focus of research around the world [1]–[7]. The various concepts of skills is quite different, but most of them are composed of primitive, formal descriptions of sets of robot motions, called action or motion *primitives*. These primitives are simple, atomic robot movements that can be combined to form more complex behavior [8]–[11], which is often called a robot *skill*. As such, no single, unified definition of a skill in terms of robotics exists. In this paper, we present our model of a robot skill, that has been previously discussed in [12], and show how these skills can be implemented and used, using a few skills as examples.

In contrast to the skills themselves, the *skill primitives* [5]–[7] are rather well defined in the robotics community; although several different descriptions exist, most of them loosely follow Mason's work on compliant controllers [13], which paved the ground for the Task Frame Formalism (TFF) introduced by Bruyninckx and De Schutter [14], [15]. Recent work has expanded upon this idea, for instance enabling the use of any sensor and controller in the same task frame (e.g. visual servoing combined with force control), and operating in a dynamic task frame [16]–[18].

The concept of skills presented in this paper is comparable to the *manipulation primitive nets*, described best in [17]. These nets are sequences of *manipulation primitives*, with simple decisions based on the outcome of each single manipulation primitive in the net. In these nets, however, each manipulation primitive needs its own explicitly specified parameters, e.g. a specific force or velocity in a specific direction. This makes this particular implementation unsuitable for robotics novices in a factory hall. Instead, we propose a method for specifying parameters on a higher level, the skill level, and instead letting the system infer the parameters for the lower level primitives.

In this paper we will show our skill paradigm applied for two of the most important robot skills for manipulation - the *pick up* and *place* skills. These are very complex skills, in that they require a high degree of both sensing and motion to be sufficiently general. Being sufficiently general, they can also be used in almost any task, be it machine tending, part feeding, etc. In order to further show the capabilities of our approach, we show that the skills can easily be ported to a totally different robot, and can be sequenced to program a robot task. As our experimental platforms we use a commercially available PR2 robot and a custom industrial mobile manipulator, the Little Helper.

In section II, we introduce the reader to the concept of skills used in this work, along with a brief explanation of the skill primitives. In section III, we will present the outline of two skills, as well as the necessary skill primitives for these. After this, we present the implementation of these skills on two very different robots in section IV. We will briefly present the roadmap for future work and conclude this paper in sections V and VI.

[1]Department of Mechanical and Manufacturing Engineering, Aalborg University Copenhagen, 2450 Copenhagen, Denmark

[2]School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30318, USA

II. Conceptual overview of skills and tasks

We will now introduce the definitions of skills and tasks used in this work. We present a layered structure, where tasks are the highest abstraction layer, and also the layer that is easiest to understand for robotics novices. Each task is composed of a sequence of skills, each of which is composed of a number of skill primitives, where the primitive layer is the most robot-oriented layer. See also Fig. 1. We will start by explaining the concept of a task.

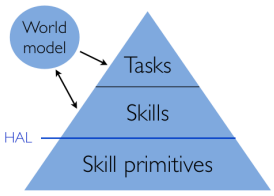


Fig. 1: The three layers of the skill model. The skills can access and alter information in the world model, and the tasks can only access the information, and alters it through the skills. The only hardware-specific parts are the skill primitives, so the hardware abstraction layer (HAL) is between the primitives and the skills.

We define tasks as sequences of robot skills, and the sequences can either be pre-programmed by an operator or generated by a task-level planner, either offline or ad hoc. Tasks are characterized by a relation to a production-related goal in the factory; for instance a machine tending task could be *insert part P in machine M, start the machine, remove part, place on workpiece tray*. The basis for a given task is a set of known state variables, such as locations of known objects, current state of production equipment, etc. The task uses skills to change these state variables to a desired goal setting.

We define a robot skill as a fundamental software building block, that incorporates both sensing and action. In order for the skills to be useful for task-level programming, they must both change the world state through sensing or action [7], and be self-sustained, so they can be used in any task. Self-sustainability implies that each skill should be

- parametric in its execution, so it will perform the same basic operation regardless of input parameter,
- able to estimate if the skill can be executed based on the input parameter and world state, and
- able to verify whether or not it was executed successfully.

A graphical representation of a skill, implementing all of these aspects, is shown in Fig. 2.

The skill uses the current world state as input, along with a parameter, which is provided at task programming time. As an example, for the *pick object* skill the parameter is the specific object that needs to be picked up. The skill initially checks that all preconditions for executing the skill are satisfied, based on the inputs. If true, the skill executes the sequence of skill primitives that changes the world state to the desired goal setting. After execution, it is verified that

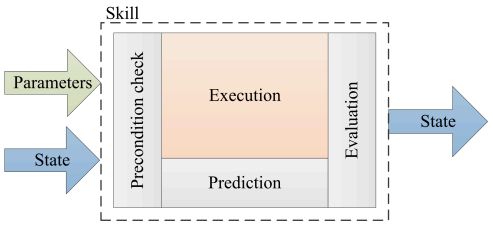


Fig. 2: Model of a robot skill

the current, measured state variables are satisfactory. This is done by comparing them to the *prediction* of the outcome (see Fig. 2), which is established from the parameter input and initial state setting.

Pre- and postconditions are required for both robustness and planning. There is no reason to attempt to execute a skill if the preconditions are not satisfied, and the skill is only correctly executed if the postconditions are satisfied. Furthermore, if there exists a task-level planner that is able to create tasks as sequences of skills, the same planner would be able to deal with precondition failures, by planning a sequence of skills that satisfies the preconditions. In the case of postcondition failures, however, the case is not as trivial, since a planner would still utilize the same library of skills, resulting in a similar task as the one currently being executed. In this case it might be feasible to simply retry the skill a few times, call an operator, or render the skill invalid for the next planning attempt.

Using these definitions of tasks and skills, and based on our findings in the FP7 project TAPAS and numerous experiments on the shop floor, a total of 566 manual tasks in a production facility have been analyzed [19], and we have shown that it is possible to break down most of these tasks into a set of only 13 distinct robot skills, and even less in simpler domains [12].

A. Skill primitives

In order for the robot skills to be robust, hardware-abstracted and easy to implement for robot skill developers, they must themselves be composed of lower-level skill primitives. We will not attempt to provide a strict definition of what a skill primitive is and what it is not. Instead, we see skill primitives as traditional macros, that perform a basic function directly which may be readily provided by the robot system. Comparing to skill primitives in humans, they are intuitively the motions and sensing we perform without thinking further about it. As such, a robot skill primitive could be as rudimentary as closing the gripper on the robot, or as advanced as planning and performing collision-free motion of a robot arm.

Being the only interface to the hardware in this model, these components are also effectively the hardware abstraction layer. Ideally, this means that any single skill primitive can be changed, as long as the same interfaces are maintained.

It is important to note that, for the sake of generality, we do not distinguish between skill primitives that purely perform motions and primitives that purely perform sensing.

For example, both an object detector and a point-to-point robot arm motion are skill primitives.

We will now go on to present the implementation of the pick and place skills.

III. Implementation overview

This section will present the challenges in implementing the pick and place skills, which is the initial implementation of skills following the concepts mentioned above. As such, we divide this section into two subsections. First, we will briefly present the outline and contents of both skills, following the model introduced in II. After this, we present a general overview of both skills, along with aspects of the implementation that is necessary for both. In this section we will also introduce the implementation of the skill primitives that are the same for both robots.

A. Skill descriptions

The pick skill aims to fulfill the goal of picking up a *known* object in the world model. The skill is designed to assume that the object is within reach of the robot, without the need of driving to the object location. Knowing that the object is within reach, the robot will re-detect the object, to get an accurate and up to date object pose. After this, it will pick up the object, and lift it away from the table. This ends the execution phase of the skill. It will then need to be verified that the execution was successful, which is done by *a)* checking if an object is in the gripper, and *b)* no object is at the previous location. The steps involved are shown in Table I, and the skill primitives used are marked with the abbreviation *SP*.

TABLE I: Pick skill

Parameter	Previously detected object
Preconditions	Object known in world model Object within reach Gripper empty
Execution	*SP:* Redetect and update pose of object *SP:* Open gripper Calculate pregrasp and grasp pose *SP:* Planned, collision-free arm motion to pregrasp pose *SP:* Motion into grasp pose *SP:* Close gripper *SP:* Motion away from table
Postconditions	Object in gripper Object not at previous location

Similarly to the pick skill, the place skill assumes the position to place the object is within reach of the robot arm. The goal of the place skill in this implementation is to put down the object in the gripper at a specific location on a known surface. A robust and general place skill, that can handle advanced contact definitions, e.g. assembly tasks, is out of scope of this particular work. The robot will have to move the gripper to a position above the desired final position of the object, after which it will move down, normal to the table, until contact. After this, it will open the gripper and move it away from the object, which concludes the execution phase. Finally, it is verified that the gripper is empty, and the object is placed at the desired position. The steps involved in the place skill are shown in Table II.

TABLE II: Place skill

Parameter	(x, y) location on previously detected surface
Preconditions	Surface known in world model Place location within reach Object in gripper Place location empty
Execution	Calculate pre-place and place poses *SP:* Planned, collision-free arm motion to *SP:* Motion down to contact with surface *SP:* Open gripper *SP:* Motion away from object
Postconditions	Gripper empty Object at correct location Gripper is away from object

B. General implementation

Certain prerequisites in the form of skill primitives need to be satisfied in order to implement the skills on the robots. Besides simple motion commands, such as gripper motions, the relatively advanced skill primitives necessary for this implementation are:

- Collision-free arm navigation
- Object detection, recognition and grasp planning
- Cartesian motions with force feedback

Furthermore, it is necessary to have a suitable representation of the world state, which we have implemented as well. The world model in this initial implementation contains information about previously detected objects and surfaces, in global coordinates. This implementation also enables advanced querying (e.g. *return objects currently located on table 1*) and updating poses of surfaces and objects ad hoc. When starting up the robot, we perform an initial scan of the scene, and save information regarding objects and surfaces, or load a previously saved world state.

The current implementation relies heavily on available ROS packages for the primitives. However, due to the layered architecture, the primitive layer could in principle be any other software structure, that facilitates a package-like structure and a well-defined communication framework.

For the object and surface detection skill primitive, we use the ROS package `tabletop_object_detector`, which performs tabletop segmentation, object detection and object recognition based on measured point clouds of the environment, captured with a Kinect camera. This package is furthermore utilizing a database of known objects, with a set of simulated grasp poses for the PR2 parallel gripper for each object. For the purpose of example, this package is adequate, and can easily be exchanged with a more suitable primitive when needed.

In order for the robot to safely manipulate objects in semi-structured environments, such as production facilities, a skill primitive that performs collision-free arm motions is also required. In order to do this, we use the ROS stack `arm_navigation`, that is fully integrated to work with the PR2.

It is not always feasible to plan every motion of a robot arm, especially not during manipulation of parts for which we do not have a model we can input in the collision map. Therefore, a skill primitive that enables motions according to the Task Frame Formalism has been implemented. This primitive is a layer on top of a low level controller, that controls the joints of the robot arm directly. Therefore, the TFF primitive is not performing real-time control of the joints of the robot arm directly, and is therefore strictly not a TFF controller. However, this primitive does enable us to specify the desired position, velocity or applied force in any direction and in any known frame, until a certain stop criterion is met, e.g. a traveled distance, a velocity, or an external force in a particular direction.

We will give a more detailed description of the implementation in the next sections.

IV. Implementation on two robots

We begin by presenting the implementation on the PR2 robot, which has essentially served as an initial prototyping platform for the skill concept introduced in II. In this description, we will present how each skill primitive was implemented, and how it fits into the skill itself. Finally, we describe the necessary steps in order to transfer this implementation from the research platform, that the PR2 is, to an entirely different robot, the industrial mobile robot Little Helper.

The skills have all been implemented as classes in Python, that inherit from a base skill class. This base class contains methods for adding pre- or postconditions, as well as evaluating all conditions. It also contains an empty method for interfacing to the ROS topics and actions, that are used in the specific skill implementation, and an empty method for execution. When implementing a skill, the programmer must overwrite these two functions with skill-specific ones, as well as add pre- and postconditions to the skill. In this way, it is ensured that all skills maintain the same vocabulary for calling the functions within the skill.

A. Implementation on the PR2

The following two sections provide a detailed description of the implementation of the skills, according to the outline in Table I and II, respectively.

1) Pick skill: As a unique ID identifies each object in the world model, this ID is used as the parameter for the pick skill. The world model is queried for the information regarding this particular object, e.g. the type of object, pose, and surface it is resting upon. The preconditions *object within reach* and *object known in world model* can be determined based on this query. The precondition check for the gripper being empty is tested on the internal joint state of the robot. If any of the preconditions fail, the skill terminates and outputs which precondition(s) failed.

In the execution phase, it is first necessary to redetect the scene by calling the object detector primitive, to accurately update the object pose in the robot coordinate system. This is due to the fact that the world model for a mobile robot will be populated with object poses in world coordinates, and current navigation and localization are not sufficiently precise for manipulation.

The grasp pose is selected based on a series of presimulated grasps, available for the specific object type in the database of objects. The chosen grasp is a selected as a tradeoff between the pose difference between the current gripper pose and the available grasp poses, and the success probability from grasp simulations. The pre-grasp pose is similar to the grasp pose, with an offset in the gripper frame, to ensure a collision-free approach to the object.

The end-effector is moved to the pre-grasp pose, by using the arm navigation primitive. The planner used is the SBL planner [20] from the Open Motion Planning Library [21].

In order to approach the object, the robot makes use of the TFF skill primitive, and performs a TFF motion into the grasp pose. This motion is specified as a velocity in the end-effector frame, towards the grasp pose, and the stop criterion is either the approach distance used to calculate the pre-grasp pose, or when sensing a contact with the object. The TFF primitive outputs desired positions to a cartesian position controller with force and velocity feedback, the `JTTaskController` in the `pr2_manipulation_controllers` package. A separate PI control of velocity and force has been implemented, where a desired pose is sent to the cartesian controller, based on the velocity or force feedback directly from the controller, respectively. After the motion reaches the stop criterion, the gripper is closed to grasp the object. The pose of the object in the world model is then updated to the current pose in the gripper frame, and a flag is set to indicate that this object is currently in the gripper. The collision model of the object is also attached to the arm, to enable future planned motions with the object in the gripper.

The TFF motion performing the lift is also a velocity command, with the stop criterion being a pre-programmed lifting height. This motion is carried out in a direction normal to the surface, to ensure there is no collision with the surface.

If any of the previous steps fails, the skill terminates and outputs at exactly which step it failed, and for what reason.

After the execution the postconditions are verified. Checking if the object is in the gripper is tested on the current gripper joint configuration, and comparing this to the known joint configuration for the chosen grasp. The detection primitive is called again, in order to ensure no objects are present at the location the object was picked from. If both postconditions are satisfied, the skill is concluded, and the robot returns to a waiting state.

2) Place skill: Similar to the objects, detected surfaces are identified in the world model with a unique ID. The parameter to the place skill is therefore the ID of the surface, and the (x, y) location to place the object on that surface. In this implementation, the coordinate is explicitly specified when calling the skill, but initial experiments with simple gesture recognition for parameter input show that this is a feasible approach [22]. The first precondition checks are whether or not the surface is known in the world model, and

if the desired location is within the workspace of the robot, similar as the checks in the *pick* skill. The world model is also queried to obtain knowledge of the object in the gripper, based on the flag set after picking up the object. Calling the object detection primitive reveals whether or not there already is an object at the specified location, in which case that particular precondition fails.

The execution phase initially calculates the place pose from the (x, y) location in the table frame, specified in the parameter. This is the desired final pose of the object, and since the object pose in the gripper is known from the world model, the final pose of the gripper is easily deduced. The pre-place pose is an offset of the place pose in the direction normal to the surface.

We again use the arm navigation primitive to conduct a planned motion to the pre-place pose. In this way we can still avoid collision with both the arm and the object in the gripper, since the object was added to the collision model of the arm. Upon reaching the pre-place pose, the TFF primitive is used to move the grasped object toward the surface, in the direction of the surface normal. This motion is terminated once a contact is sensed with the table, and the gripper is opened. The collision model of the object is now detached from the arm, as it is no longer needed.

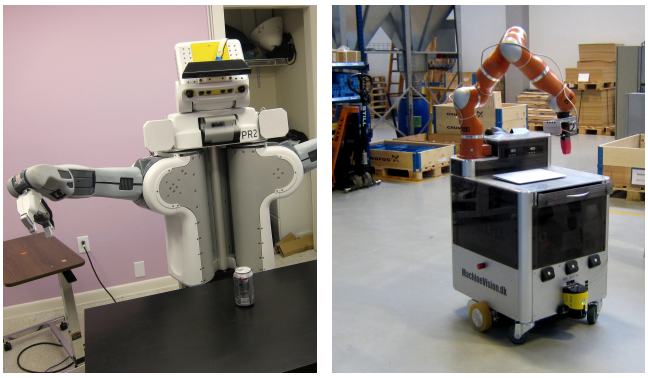
In order to avoid colliding with the object when moving away from it, this is also carried out as a TFF motion in the gripper frame, similar to the approach motion in the pick skill. The stop condition is a distance greater than the length of the gripper fingers, to ensure the gripper is clear from the object.

After the gripper is moved away, the pose of the object is updated to be the desired place pose. This is then verified as a post-condition by re-detecting the scene, and if the object is within an acceptable placing tolerance, the pose of the object is updated to be the actual, detected pose. The checks if the gripper is empty (fully open) and away from the object are measured based on the internal robot state.

B. Implementation on the Little Helper

The Little Helper (see Fig. 3) relies on a KUKA Light Weight Robot (LWR) arm for manipulation, which is not fully compatible with ROS. Furthermore, the system consists of a Schunk WSG-50 parallel gripper for manipulation, and a Neobotix MP-L655 mobile differential drive platform for mobility. Both of these latter components are fully integrated with ROS.

The primary tasks in porting the skills to the Little Helper are to implement missing primitives, modifying the interfaces with the primitives, and to make adjustments in the various geometric calculations within the skill. This is primarily calculating gripper poses, but also mapping the joint configurations of a grasp with the PR2 gripper to that of the gripper on the Little Helper. Considering nearly no attention to hardware abstraction was taken when initially developing the skills on the PR2, the skills were working on the Little Helper within 1-2 days and with low effort. If



(a) (b)

Fig. 3: The two robots used for experiments: (a) the PR2 robot and (b) the Little Helper robot at a production facility

the skills had initially been developed in a generic manner, it would have taken even less time and work.

On the Little Helper, we are using a skill primitive that sends cartesian or joint poses directly to the KUKA controller, which then handles inverse kinematics, path planning and joint interpolation. This has the drawback that it is not possible to perform collision-free arm navigation and TFF-like motions. It is, however, very important to again stress the fact that the interfaces to the different primitives are the same. So when the arm navigation and TFF skill primitives are working on the Little Helper, they can directly be used by the skills. This shows how easily skill primitives can be exchanged, as long as their interfaces are the same.

In the initial experiments with the PR2, we have used the robot in a stationary position. On the Little Helper, however, we have implemented one more skill, which is the simple *drive to station* skill. This skill uses the ROS navigation system to navigate the mobile base in a known map to a workstation that is previously saved in the world model. This skill only has the precondition check of whether or not the station is known in the model. The skill will then call the navigation primitive, in order to bring the mobile robot from the current position to the saved workstation, and check that the robot is at the goal.

Using these three skills, we can easily create small scripts, that is essentially task descriptions. In the following is shown a simple function that calls the skills to get a box of rotor caps, for rotors used in a domestic water pump, and transport them to a rotor press, where the final rotor is assembled:

```
def get_box_of_rotor_caps():
  # Drive to rotor cap press
  drive_to_station('warehouse')
  # Get ID of a box of rotors near the robot
  box_id = get_object_id('full_rotor_box',near='warehouse')
  # Pick up a box of rotors
  pick_up(box_id)
  # Place on robot platform at a specific position
  place('platform',0.30,0.55)
  # Drive to the rotor assembly station
  drive_to_station('rotor_press_station')
  # Pick up the box from the platform
  pick_up(box_id)
  # Place the box on the table next to the press
  place('rotor_press_table',1.40,0.25)
  return
```

This is only the very basic way of using skills to program

tasks, and being able to specify robot tasks as simple as the example above opens up the door to a lot of different possibilities in Human-Robot Interaction, that can output scripts like this. Recent experiments include both gesture input for the parameters, and specifying the sequence of skills in a GUI.

V. Future work

It is our intention to further develop the library of skills, with the 10 (less advanced) skills that are missing for the logistic domain described in [12]. We will do this only on the Little Helper. When we have a sufficient library of skills, we will use them for improved human-robot interaction, so an operator can select the sequence of skills, and their parameters, in a GUI or through direct interaction with the robot. It is imperative that this will be extensively tested in a real production facility. Furthermore, we will experiment with using simple planners to create sequences of skills, and more advanced ones to deal with skill failures, both as pre- and postcondition failures, and during execution.

VI. Conclusion

In this paper we have presented a model for programming robust, hardware-abstracted robot skills, for use in task-level programming, following a structure similar to how we humans use basic skills to form more complex tasks. One can argue that the implementation has a number of shortcomings, such as requiring a database of known objects with corresponding grasps. However, we would like to argue that the modular concept of the skills allows to very easily change such components to accommodate the needs of the different scenarios. Our skill model can easily accommodate future extensions, both in skill primitives and world state knowledge. In principle, this concept looks very promising, because the skills are self-sustained, in that they know when they fail and why, and are parametric in their execution. But also because the skills are highly modular, since they are composed of lower level skill primitives. Getting a skill, explicitly designed for one robot, to work on another robot proved to be effortless, given the right skill primitives. However, having the right skill primitives readily available can be the limiting factor in a skill implementation, if the skill is to be implemented in the exact same manner on all robots.

For demonstrations of the described implementations, see the attached video.

ACKNOWLEDGMENT

The authors would like to thank the Center for Robotics and Intelligent Machines (RIM) at Georgia Institute of Technology for the opportunity of using the PR2 for experiments. Especially, we would like to thank Tucker Hermans from RIM for always being helpful.

This work has partly been supported by the European Commission under grant agreement number FP7-260026-TAPAS.

References

[1] M. Lopes and J. Santos-Victor, "A developmental roadmap for learning by imitation in robots," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 37, pp. 308–321, Apr. 2007.

[2] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, pp. 14 – 23, Mar. 1986.

[3] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, p. 481–487, 2002.

[4] R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2, p. 109–116, 2004.

[5] S. Schaal, "Is imitation learning the route to humanoid robots?," *TRENDS IN COGNITIVE SCIENCES*, vol. 3, no. 6, p. 10, 1999.

[6] A. Björkelund, L. Edstrom, M. Haage, J. Malec, K. Nilsson, P. Nugues, S. Robertz, D. Storkle, A. Blomdell, R. Johansson, M. Linderoth, A. Nilsson, A. Robertsson, A. Stolt, and H. Bruyninckx, "On the integration of skilled robot motions for productivity in manufacturing," in *Assembly and Manufacturing (ISAM), 2011 IEEE International Symposium on*, pp. 1 –9, May 2011.

[7] N. Krüger, J. Piater, F. Wörgötter, C. Geib, R. Petrick, M. Steedman, A. Ude, T. Asfour, D. Kraft, D. Omrcen, *et al.*, "A formal definition of object-action complexes and examples at different levels of the processing hierarchy," *PACO-PLUS Technical Report, available fro m http://www. paco-plus. org*, 2009.

[8] T. B. Moeslund, A. Hilton, V. Krüger, and L. Sigal, *Visual Analysis of Humans: Looking at People*. Springer-Verlag New York Inc, 1 ed., 2011.

[9] V. Krüger, D. Kragic, A. Ude, and C. Geib, "The meaning of action: a review on action recognition and mapping," *Advanced Robotics*, vol. 21, no. 13, p. 1473–1501, 2007.

[10] A. F. Bobick, "Movement, activity and action: the role of knowledge in the perception of motion.," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 352, pp. 1257–1265, Aug. 1997. PMID: 9304692 PMCID: 1692010.

[11] G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi, "Premotor cortex and the recognition of motor actions," *Brain Research. Cognitive Brain Research*, vol. 3, pp. 131–141, Mar. 1996. PMID: 8713554.

[12] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?," in *Proceedings of the 43rd International Symposium on Robotics (ISR)*, (Taipei, Taiwan), Aug. 2012.

[13] M. T. Mason, "Compliance and force control for computer controlled manipulators," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 11, pp. 418 –432, June 1981.

[14] H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the "task frame formalism"-a synthesis," *Robotics and Automation, IEEE Transactions on*, vol. 12, pp. 581 –589, Aug. 1996.

[15] J. De Schutter and H. Van Brussel, "Compliant robot motion i. a formalism for specifying compliant motion tasks," *The International Journal of Robotics Research*, vol. 7, pp. 3–17, Aug. 1988.

[16] T. Kröger, B. Finkemeyer, and F. Wahl, "Manipulation primitives — a universal interface between sensor-based motion control and robot programming," in *Robotic Systems for Handling and Assembly* (D. Schütz and F. Wahl, eds.), vol. 67 of *Springer Tracts in Advanced Robotics*, pp. 293–313, Springer Berlin / Heidelberg, 2011.

[17] B. Finkemeyer, T. Kröger, and F. M. Wahl, "Executing assembly tasks specified by manipulation primitive nets," *Advanced Robotics*, vol. 19, pp. 591–611, June 2005.

[18] T. Kröger, B. Finkemeyer, S. Winkelbach, L.-O. Eble, S. Molkenstruck, and F. Wahl, "A manipulator plays jenga," *Robotics Automation Magazine, IEEE*, vol. 15, pp. 79 –84, Sept. 2008.

[19] S. Bøgh, M. Hvilshøj, M. Kristiansen, and O. Madsen, "Identifying and evaluating suitable tasks for autonomous industrial mobile manipulators (AIMM)," *The International Journal of Advanced Manufacturing Technology*, vol. 61, pp. 713–726, July 2012.

[20] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics Research* (R. A. Jarvis and A. Zelinsky, eds.), vol. 6, pp. 403–417, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[21] Kavraki, Lydia E., "The open motion planning library (OMPL)," 2010.

[22] M. R. Pedersen, C. Høilund, and V. Krüger, "Using human gestures and generic skills to instruct a mobile robot arm in a feeder filling scenario," in *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA)*, (Chengdu, Sichuan, China), Aug. 2012.