A New Framework for Task Oriented Sensor Based Robot Programming and Verification

K. Diethers and T. Firley
Program Development Dept.
Institute of Software
Mühlendortstr. 23
D-38106 Braunschweig
{diethers,firley}@ips.cs.tu-bs.de

T. Kröger and U. Thomas
Institute for Robotics
and Process Control
Mühlendortstr. 23
D-38106 Braunschweig
{t.kroeger,u.thomas}@tu-bs.de

Abstract
We evaluate the usage of skill primitive nets for programming robots, which support hybrid control in a uniform way. This notation gives an easy and intuitive way to switch between position, velocity, and force control for each DOF independently as long as no inherent dependencies are affected.

Robot specific restrictions within the work space are an intrinsic problem of the approach to a portable programming interface. We deal with this problem by using the model checker HyTech to analyze a robot program consisting of skill primitives. This analysis verifies nets of skill primitives under consideration of these restrictions.

1 Introduction
Modern and easy to use programming concepts for industrial robots still are subject in research projects. In literature, various programming methods exist from simple move commands to complex task-oriented commands like Move block A on top of Block B as it is implemented in the early R-APT system [1]. Most of these programming concepts are restricted to position and velocity move commands only and do not employ external sensors. In the past decades, many commercial robot programming languages like AML, V+, and BAPS have been developed. Incorporation of external sensors in these interpreter languages is treated weakly or not at all. Usually robots interact with their environment. Considering uncertainties, the incorporation of sensors in modern programming concepts is indispensable. Some of the mostly in research labs developed languages are embedded in universal programming languages like PasPo in Pascal, RCCL in C and Zero++ in C++ [2,3,4]. With these programming languages, sensor based motion commands are enabled. Nevertheless, writing robot programs still is a difficult, time and cost consuming part of the industrial automation process. For example, singularities and restrictions due to joint ranges also have to be taken into the programmer’s consideration, while implementing robot tasks.

Since Mason’s compliance frame concept, we know how to express sensor based move commands for robots in an intuitive manner [5]. This classical paper has been confirmed by Raibert and Craig [6] in the early eighties. In the past, many robot controlling schemes for hybrid position/velocity and force control have been introduced, e.g. [7,8,9]. The above mentioned robot programming languages do not take benefit of these hybrid concepts. Therefore, in [10] we have introduced our robot programming methodology of skill primitive nets. Fig. 1 illustrates the skill primitive concept as interface to different robot control architectures for programming various types of robots.

The programmer is able to think in robot tasks, and

![Figure 1: Skill primitives as general framework for programming robots kinematic-independently](image-url)
all controlling and kinematic details are hidden behind a well defined interface. Even the difference between programming serial or parallel robots diminishes. The skill primitive interface also suited very well as general interface for further planning concepts, rudimentarily outlined in [10].

In order to protect robots and their environment, skill primitive nets have to be analyzed prior to execution. Applying formal verification techniques allows us to investigate deadlocks, states, which are not reachable, occurrences of singular positions at runtime, etc. It supports the program developer while coding complex robot tasks in skill primitive nets.

Formal methods like theorem proving or model checking are used in industrial projects to analyze systems thoroughly and mathematically exactly. Especially model checking is well suited to verify formal models, since the available tool support enables automatic verification, and an error trace gives valuable information for correcting erroneous models. We chose the model checker HyTech, because it supports hybrid automata as formal model, which is necessary to model continuous values of physical properties.

1.1 Related Work

In [11] Petri nets are applied to describe robot tasks, and in [12] statecharts are used to write robot programs for force guided motions. They apply HyTech to analyze statechart diagrams. In contrast to their approach, our motion commands are based on higher granularity, and each skill primitive is a hybrid controlled motion instead of a force guided motion.

Firstly, we introduce the skill primitive concept, followed by a description of how to analyze this concept by applying HyTech as model checking tool. We investigate a skill primitive example in order to prove the absence of certain errors.

2 Task Oriented Robot Programming Using Skill Primitive Nets

With the programming methodology of skill primitive, the end-user is enabled to think in robot tasks like: shaft of screw shall fit into hole of another workpiece; place block on table; secure light bulb in bayonet socket. These tasks are implemented by a program developer, who applies the skill primitive concept. The end-user can combine predefined robot tasks in few intuitive steps instead of writing many single move commands.

2.1 Definition of Skill Primitive Nets

Each single skill primitive is composed of a compliance frame $C$, a stop condition $\lambda$, and a declaration of the Center of Compliance $\text{CoC}$, i.e. a frame, where the compliance frame is referenced to. Hence, all the programming is done in the task frame formalism, also mentioned in [7]. For hybrid force/velocity or position control, we follow Mason’s concept. For instance, the skill primitive

$$\text{diag}(C) = (0\text{mm}, 0\text{mm}, 3\text{mm/s}, \text{0 rad}, \text{0 rad}, \text{0 rad})^T,$$

$$\lambda = (F_z \leq -10\text{N}) \vee (\text{time} \geq 5\text{s}).$$

means a movement in $z$-direction until either a force threshold value of $-10\text{N}$ is reached or a timeout is triggered after more than 5 seconds.

Complex robot tasks can be described as a sequence of such skill primitives or a net of skill primitives, where a certain path through the net is selected in dependence on measured sensor values at runtime.

Formally, a skill primitive net $SPN := \{\Sigma, \Pi, \Xi, \Upsilon, \Omega\}$ is defined as follows: $\Sigma$ is a set of all skill primitives. $\Pi$ is a finite set of start states. $\Xi$ is a finite set of stop states. $\Upsilon$ is a finite set of error states and $\Omega$ corresponds to a set of directed edges linking the skill primitives to each other. Each edge is either attributed by an entrance condition or by the empty condition. If one skill primitive has reached its stop condition, its execution stops and the next skill primitive is evaluated. We require that the disjunction of all attributes of outgoing edges is equivalent to the skill primitive’s stop condition, so that the next skill primitive is well defined. For a formal definition, we refer to [10].

2.2 The Light Bulb Example

We have coded some complex robot tasks by means of skill primitive nets. The light bulb example suites well to outline the approach and to demonstrate model checking aspects. Fig. 2 shows the light bulb to be secured in a bayonet socket. Assuming, the rotational axis of the bulb is fairly aligned with the axis of the socket. Execution of this robot task can be split into

![Figure 2: The light bulb and the bayonet socket (left), the corresponding configuration space $(z, \varphi_z)$ (right)](image-url)
movements, which are illustrated on the right hand side of Fig. 2.

In Fig. 3, a corresponding skill primitive net for the robot task is shown. The first skill primitive (I) brings the light bulb in contact with the socket. Two traversals of the net are possible. The right branch is selected if the light bulb’s pins are aligned to the notch. In the other case, the bulb’s pins (only one is depicted in Fig. 2) do not fit into the notch. In the other case, the bulb’s pins are aligned to the notch. This is achieved by applying skill primitive II. The light bulb is turned on top of the socket until the force changes by more than $-5 \text{ N}$, the position of $P_z$ changes by more than $3 \text{ mm}$, and the 50 milliseconds-average value of the torque $T_z$ equals 0 Nm. By the next skill primitive (III), the light bulb moves further down until a force value of $-20 \text{ N}$ in $z$-direction is passed and the spring is compressed. Now the light bulb is rotated around the $z$-axis, until a torque value of $-0.2 \text{ Nm}$ can be measured. This is achieved by skill primitive IV. Skill primitive V conjoins the two traversals. To secure the light bulb, a movement in negative $z$-direction is necessary until the force arises to more than $5 \text{ N}$.

\[ \text{diag } C = (0 \text{ mm}, 0 \text{ mm}, 3 \text{ mm/s}, 0 \text{ rad}, 0 \text{ rad}, 0 \text{ rad}) \]
\[ \lambda : \{F_z \leq -20 \text{ N} \} \]

\[ \text{diag } C = (0 \text{ N}, 0 \text{ N}, -25 \text{ N}, 0 \text{ Nm}, 0 \text{ Nm}, -0.5 \text{ Nm}) \]
\[ \lambda : \{ ((F_z \leq -15 \text{ N}) \bigwedge (T_z \leq -0.2 \text{ Nm})) \}
\[ \bigvee ((F_z \geq -5 \text{ N}) \bigwedge (P_z \geq 3 \text{ mm}) \bigwedge (T_z = 0 \text{ Nm}^{15 \text{ ms}})) \} \]

\[ (F_z \geq -5 \text{ N}) \bigwedge (P_z \geq 3 \text{ mm}) \bigwedge (T_z = 0 \text{ Nm}^{50 \text{ ms}}) \]

\[ \text{diag } C = (0 \text{ N}, 0 \text{ N}, 3 \text{ mm/s}, 0 \text{ Nm}, 0 \text{ Nm}, 0 \text{ Nm}) \]
\[ \lambda : \{F_z \leq -20 \text{ N} \} \]

\[ \text{diag } C = (0 \text{ N}, 0 \text{ N}, -25 \text{ N}, 0 \text{ Nm}, 0 \text{ Nm}, -0.5 \text{ Nm}) \]
\[ \lambda : \{T_z \leq -0.2 \text{ Nm} \} \]

\[ (F_z \leq -15 \text{ N}) \bigwedge (T_z \leq -0.2 \text{ Nm}) \]

\[ \text{diag } C = (0 \text{ N}, 0 \text{ N}, -3 \text{ mm/s}, 0 \text{ Nm}, 0 \text{ Nm}, 0 \text{ Nm}) \]
\[ \lambda : \{F_z \geq 5 \text{ N} \} \]

Figure 3: A skill primitive net for a light bulb to be inserted into a bayonet socket

2.3 Interface to Different Control Architectures

The skill primitive concept is supposed to be a hardware-independent interface to any robot control system. The PC-based robot control architecture of Fig. 4 has been developed to fulfill the requirements of easy hardware and software replacements. Except the block ‘Robot and sensors’, each block represents a single process running on a real-time platform. The complete interprocess communication is managed by MiRPA-X (Middleware for Robotic and Process Control Applications—Extended), which provides a very flexible and modular process environment [13]. A single skill primitive, selected out of a skill primitive net, is sent to the skill primitive execution process via MiRPA-X. According to the outputs of the on-line trajectory planner and the force/torque controller, a new Cartesian robot position is calculated. The trajectory planner is responsible for position and for velocity controlled DOFs; the force/torque controller generates a new Cartesian position for all force/torque controlled DOFs. The skill primitive concept requires hybrid control with respect to any individual task frame (CoC), hence, all respective control values are transformed into this frame. After the stop condition check, the kinematic robot model is applied to compute a new joint position, which is sent via MiRPA-X to the joint position controller. The controller output directly is connected to the robot driver. If a skill primitive stop condition is evaluated to true, the robot task process receives an acknowledgement message from the skill primitive execution process. The

![Figure 4: Applied robot control architecture](image-url)
selection of the next skill primitive depends on the data of this message. To apply the same robot task, i.e. the same skill primitive net, to another robot, only the four black-colored boxes of Fig. 4 have to be substituted. In correspondence to the new robot, the robot driver, the joint position controller, the force/torque controller, and the kinematic model have to be replaced. To embed a new force/torque sensor, the force/torque sensor driver has to be renewed either. This way, it does not matter, what kind of hardware does execute the skill primitive net. Serial robots might be substituted by parallel ones, single sensors can be exchanged or added to the system. In future projects, a distance sensor or a computer vision system might be interrogated by the hybrid controller. As result, an open system offering multi-sensor integration possibilities for future extensions has been created.

3 Analyzing Skill Primitives by a Hybrid Model Checker

In computer science, formal methods are used to prove the correctness of complex systems. However, their main advantage is to find errors. This is particularly true for the application introduced here, since formal methods work on a model of the real system. Within the model, we have to accept certain simplifications. However, we are able to analyze some properties of the skill primitive net in the theoretical model. This is useful in two cases. First, we can search for errors that are hard to find by simulation or testing. Second, we can analyze the net for different robots almost automatically, especially in early stages of the development of these nets and when automated analysis is recommended, e.g. due to several different robot structures.

3.1 Model Checking

The method we have chosen is model checking [14], which is basically an exhaustive search in the state space of the model for desired properties. The model is usually an automata based description of the system, in the sense of automata theory.

The model checking technique is a method, which mathematically searches in the state space of a model for certain properties. The state space has to be finite, since the search is exhaustive. Not only every state but also every calculation path in the state space can be analyzed, even if there are infinitely many paths. Due to the high complexity, the resource consumption is enormous. Nevertheless, this technique is very successful in hardware verification and recently also in software verification.

3.2 Hybrid Automata

Due to the necessity to describe continuous variables, we cannot use simple finite automata. However, hybrid automata provide means to deal with discrete as well as continuous flow [15]. Transitions may be labeled with actions, which are synchronized between several automata. Furthermore, there are two kinds of variables. Discrete variables may be changed by actions of a transition. Analog variables may also be set by an action of a transition, but additionally change their value while time passes within a state according to the slope that is specified in an invariant. A net of these automata is used to model the behavior of the system and its environment.

HyTech is a model checker that works on hybrid automata. Properties in HyTech are expressed by giving an initial and a target state and by trying to calculate a path from the initial to the target state. We can express that a position is not reachable or that we will reach a position in any case. It is also possible to express more complicated properties by combining several of such properties.

3.3 Light Bulb Model

We have modeled the light bulb example for the analysis with HyTech. In addition to the skill primitive net, we had to model a simple version of the controller and the environment to be able to analyze some interesting properties. Since in the example only two degrees of freedom are used, we simplified the model in this respect. Hence, we get a two dimensional space. The model is structured in three parts: the environment, which describes the shape of the socket and the spring, the controller, which switches between force and velocity control and changes the position of the manipulator respectively, and the skill primitive net that issues commands to the controller (cf. Fig. 5). Note that some physical aspects had to be embedded into the automata for the environment and for the controller. Hence, the controller does not directly model the controller of a robot.

There are several variables in our model. The two degrees of freedom are the z-direction and the rotation $\varphi_z$ around the z-axis. The position in this two dimensional space is determined by the two variables $z$ and $p$. Both variables are continuous. Their derivatives with respect to time are denoted by $dz$ for velocity $\dot{z}$ and by $dp$ for angular velocity $\dot{\varphi}_z$. Furthermore, we have variables $f_z$ for the force $F_z$ and $t_p$ for the torque $T_z$. We manipulate these variables via intermediate variables, e.g. if we want to set the force, we use $f_z\text{target}$ to pass the value to the controller.

![Figure 5: Components of the model](image-url)
The values of the physical variables are changed by the controller automaton, which performs the task of emulating the respective physical aspects of the real world.

The environment is modeled by three automata. Two of them describe the shape of the socket, one is responsible for the force caused by the spring. The two automata for the shape describe the shape for two dimensions separately. In our simplified model, we assume rectangular angles. Hence, a contact with the socket only affects one dimension at a time. The shape of the socket is divided into several regions. There are regions for the faces that we find in $z$-direction, respectively in $\varphi_z$-direction, and there are regions for the space where the bulb may move. There are no regions for the solid parts of the socket. A movement that would yield a position in the solid area would give a deadlock in our model. In Fig. 6, we depict the regions for dimension $\varphi_z$ in our example.

Whenever we enter a region, which represents a face of the socket, we issue a signal, to inform the controller to change from velocity control to force control. More precisely, we indicate in which direction a contact occurred by the signals $u$ upcontact, downcontact, leftcontact or rightcontact. Whenever we lose contact, we issue the signal $z$ free or $p$ free.

For the spring, we do not issue any signals. Instead, we distinguish between a state where the spring does not have any effect and a state where it does. We increase the force $sfz$ caused by the spring linearly.

Control in our model is kept very simple. For each degree of freedom, we switch between force driven control and velocity driven control. We switch instantaneously. We do not consider the raising or the falling of the force or acceleration or deceleration of the velocity, nor do we consider inaccuracy of the real controller. The same holds for torque and angular velocity. Hence, we have two automata, one for each dimension.

The skill primitive net from the previous section can be modeled in a straightforward way. The net is translated to one automaton. Each skill primitive is translated to three states in the automaton. In Fig. 7, we have depicted the model for skill primitive V. The first of the three states serves as target for incoming transitions. With the transition to the second state, the target values for force and torque are set. The transition from the second state to the third one issues an update signal to the controller. No time may pass while being in the first two states. Velocity and angular velocity are defined by the slope of the variables $z$ and $p$ as invariants in the third state.

3.4 Verification Properties

By model checking it is possible to analyze a wide range of properties. For the skill primitive net, reasonable verification attempts include the search for unwanted positions, e.g. singular positions, the reachability of all skill primitives under consideration of the environment, possible deadlocks in the net, and successful termination of the net. It is also possible to calculate starting positions, which have to be avoided, or values, which need to be used in the skill primitives to ensure termination. For this paper, we restricted the analysis to three properties (cf. Fig. 8):

1. Successful task termination. We checked that the target region $A$ will be reached in the normal case. On the one hand, we could show that $A$ is reachable. On the other hand, we found a time bound, in which it is not possible to remain outside the target region.

2. Unsuccessful starting positions. For this analysis,
we assumed that region \( C \) is a singular position. We performed a backward analysis to calculate all positions, from which it is possible to reach \( C \). These positions have to be avoided.

3. Deadlock due to singular position. This property is similar to property 2. However, here we placed the forbidden region \( B \) in a way, which makes it impossible to avoid it.

Once the system and its environment is modeled, the analysis of different properties can be automated. It is easy to define forbidden regions, and to analyze variations of given properties.

4 Results

4.1 Analysis by HyTech

For each property, we get a trace through the automata of the model. If the property holds true, we get a witness, if it does not, we get a trace to an error state. Since the model is rather simple in some respects, we checked for every trace manually if it corresponds to a possible path in the real system.

A drawback of the current approach is the time consuming and error-prone manual construction of the formal model. This case study shows that automation of the translation is necessary and feasible.

- The part of the model, which implements the net of skill primitives is constructed in a schematic way. This part can easily be automated. Depending on the verification property under consideration, the model can be optimized. For example, if time is not important, constructs that do not affect the robot movements can be omitted to reduce the global state space.

- The controller automaton of the model encodes physical aspects. To investigate the core semantics of skill primitives, which is intended to be independent of a given implementation, this part remains unchanged. Considering more specific physical properties requires an adaption of the model. However, this is out of the scope of this paper.

- An automation of task descriptions consisting of convex regions is straightforward. If a complex model is needed, user interaction is necessary to find an appropriate abstraction.

We verified the properties mentioned above on a Sun UltraSparc. The verification of these properties takes about 1 GByte of memory and between 20 and 60 minutes time.

4.2 Execution

Independently from the formal analysis, we have implemented and tested several skill primitive nets for robot tasks like: object placing, object sliding, object mating in U/L-block. All these experiments have shown that execution of robot tasks succeeds in a stable and reliable way. We have implemented a reference control for a serial robot, which gave us valuable experience for the improvement of this concept and proves that the approach is feasible. Fig. 9 shows the execution of the robot task “Light bulb insertion into a bayonet socket”. Within our collaborative research center 562, we are working on a control for a parallel robot. In this context, we expect that our approach of model checking by HyTech eases the adoption of skill primitive nets to new robot kinematics significantly.

Figure 9: Experimental set up for light bulb insertion

For the case that these tasks are executed by parallel robots, which will be performed in future time, the software modules mentioned in section 2.3 have to be replaced. The skill primitives remain untouched. The program developer only has to care about the number of DOFs. The coding of robot tasks can be done much more efficiently. Nevertheless, without an automated analysis programmers need to be aware of kinematic details, occurrences of singularities etc.

5 Summary and Conclusions

Skill primitives constitute a general interface to hybrid position/velocity and force controlled systems for robots with arbitrary kinematic structures. Implementing robot programs with the approach of skill primitive nets is much more efficient than traditional approaches. In order to avoid erroneous executions of robot tasks, formal verification methods are essential. As demonstrated, model checking is a feasible solution.

The automatic planning of robot tasks in terms of skill primitive nets still is far from industrial application. Libraries with implementations of robot tasks could be provided in near future. For this purpose, model checking supports the program developer.
For our investigations on model checking, we took an ideal execution of skill primitives and abstracted from control specific characteristics referring to the implementation of skill primitives. We showed that model checking by HyTech can be used on this abstraction level and gives valuable analysis results. The program developer can model control specific forbidden regions and investigate their reachability. He also can check, if programs always terminate in the desired final skill primitive within a certain time bound. We demonstrated that this approach is feasible by analyzing a medium size case study.

The possibilities of automatically generating the model still have to be investigated. Especially the model of the task environment and the skill primitive net are well suited for automation. Critical aspects are the modelling of complicated environments and the consideration of physical robot properties. Here, further investigation of optimization for reducing the state space of the model is necessary. The portability of our approach is attractive to reduce costs in the industrial automation process. Within the collaborative research center 562, we will investigate the concept of skill primitives on parallel robots in the near future. Therewith, we will show, that skill primitive nets constitute a portable programming interface for robots with arbitrary kinematic structures.

Acknowledgments

We would like to thank the German Research Foundation (DFG) for supporting the collaborative research center 562.

References


