Task-Optimised Robot Kinematics

How can individual robots be efficiently designed and operated?

vorausrobotik

Carl-Buderus-Str. 7 | 30455 Hanover | Germany | vorausrobotik.com

Summary Task-optimised robots offer numerous advantages over standardised compromise-ridden kinematics: they are easier to integrate into existing systems, are optimally tailored to the process, and reduce cycle times. On the other hand, there is increased effort required in design and software engineering, which counteracts the aforementioned advantages.

In this white paper, we demonstrate how these obstacles can be overcome in practice, thereby realising significant efficiency gains. Using concrete examples, we provide a practical explanation of the design, modelling, and commissioning processes – resulting in a significant increase in efficiency and a 25% reduction in cycle time as shown for an exemplary pick-and-place application.

1. Introduction and Motivation

In September 2025, the International Federation of Robotics announced that, for the fourth consecutive year, more than half a million robots had been shipped worldwide (source: https://ifr.org/ifr-press-releases/news/global-robot-demand-in-factories-doubles-over-10-years). The trend towards (robot-assisted) automation continues unabated and is driven by a wide range of factors, such as onshoring and demographic change. Standardised kinematics from major manufacturers (hereinafter also referred to as standard robots) dominate this sector. The reasons for this include customers' existing experiences (programming, maintenance, spare parts stock, etc.) and global availability. Conversely, when selecting a specific model, compromises often have to be made regarding payload, space requirements, cycle time, etc. Task- or application-specific robots can compensate for these disadvantages – whereby the design-process can include both simple adjustments to segment lengths and axis distances (so-called dimensional synthesis) and completely new, optimised structures (so-called structural synthesis). This offers numerous advantages over standard robots, some of which are listed below: (a) Higher accuracy, dynamics, or rigidity can make processes automatable in the first place or significantly increase their efficiency – as demonstrated by the 25% reduction in cycle time shown in the case study . (b) A smaller footprint eases integration into facilities and reduces operating costs associated with floor space (air conditioning,

cleaning, building depreciation, etc.). (c) A design tailored to the payload or process avoids oversizing and reduces energy costs (as less mass needs to be moved).

Although individual robots offer significant advantages in terms of application and cost-efficiency, they have not yet become widespread. The reasons for this include the costs associated with design (including construction), control integration, and commissioning. Therefore, such solutions are (yet) primarily used in niche applications that offer correspondingly high margins or cannot otherwise be automated (efficiently) (e.g. due to the process' dynamics or precision requirements).

2. From Design to Commissioning

How can the aforementioned disadvantages of individual kinematics be mitigated and the economic advantages leveraged? To this end, we first examine the key development steps – from design (Sections 2.1 and 2.2) to commissioning (Sections 2.3 and 2.4). The relationships are illustrated in Figure 1. The upper part outlines the optimisation loop: a robot is modelled and incorporated into the simulation. This simulation includes the process and other cell components, and the application is executed virtually. Depending on the test and performance results obtained in this way, either the robot is adapted or the software modules are transferred to the real system (lower part of Figure 1).

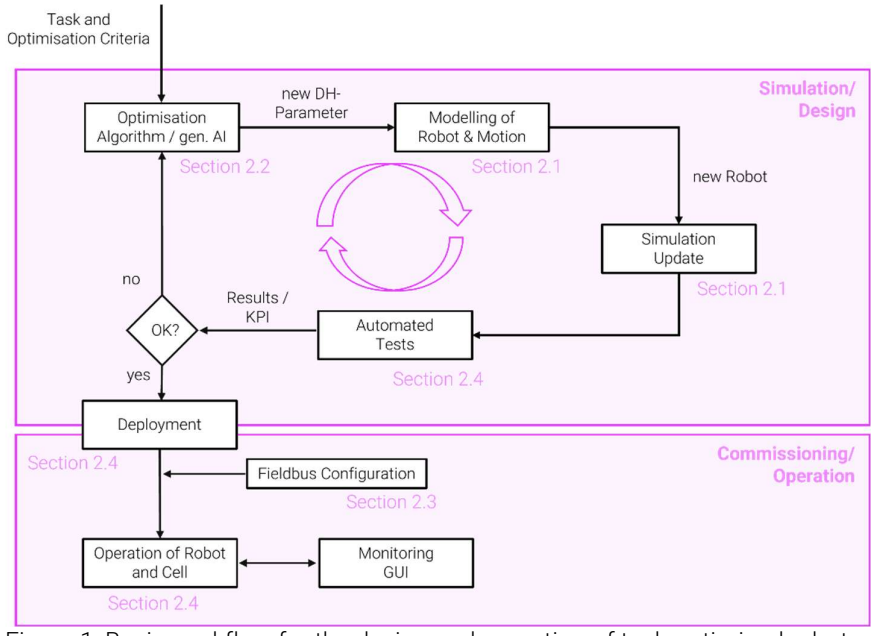


Figure 1: Basic workflow for the design and operation of task-optimised robots

2.1 Simulation, Modelling, and Application

In principle, application-specific robots can only fully realise their benefits if they are (optimally) tailored to the given task. In this context, the process and the entire system (or at least the components that interact directly with the robot) should be considered

together: changes to the robot (geometry, footprint, trajectory) affect surrounding components (accessible workspace, collisions, etc.) and the process (e.g. dynamics, cycle time). The problem is often that existing simulation tools are proprietary and only cover parts of the system. Consequently, they are only of limited use and represent a major obstacle to an efficient design process. It is also essential that the simulation environment has open interfaces, which provide the design process with access to the (virtualised) components and thus the necessary degrees of freedom for optimisation. This enables not only the optimization of the robot itself but also of the cell, such as the ideal placement of conveyor belts – if desired.

Before we consider the optimisation itself (Section 2.2), we must first address the modelling of the robot and its motion planning. It is important that the behaviour of the simulated robot corresponds as closely as possible to that of the real one – only in this way the calculated benefits can be realised in the actual implementation. The models (or, more generally, the calculations) used in the simulation must therefore also be implemented in the system's control; this holds in particular true for the path planning – the robot's movements are crucial for cycle time optimisation, collision avoidance, etc.

First, the robot's kinematics must be calculated, describing the relation between joint angles (rotational and translational) and the robot end-effector (the tip of the tool attached to the robot's flange). We must differentiate between direct and inverse kinematics: direct kinematics means that, given the joint angles, the position and orientation (together also referred to as the pose) of the robot end-effector can be calculated. Inverse kinematics solves the opposite problem: the corresponding joint angles are determined from the given pose of the end-effector. A common method for kinematic modelling relies on the Denavit-Hartenberg (DH) parameters. These describe the relative pose of the robot axes to one another in an efficient form (4 parameters per axis), thereby enabling the determination of both direct and inverse kinematics.

Based on the robot kinematics we can calculate the workspace, numerous kinematic performance criteria, and the robot's path. If we link the path (the set of all points/orientations) along which the robot moves with time, we obtain the trajectory. A wide variety of path and trajectory planning methods (e.g. linear and circular motion, smooth transitions between different path segments) are essential for efficient process control and optimisation of cycle times.

If we combine the trajectory with the robot's dynamic parameters (mass and inertia of segments, friction in the drives, etc.), we can calculate direct and inverse dynamics. Dynamics here refers to the determination of forces and torques during a movement (and, for the mechanics among us: also at rest). These values can be used for feedforward control of the drives, ensuring excellent path accuracy even at very high accelerations or payloads. Knowledge of the dynamics is also helpful when selecting the optimal motor/gearbox combination, as it provides the motor torque.

We can conclude that deriving the fundamental equations for robots requires specialist knowledge from various fields, which is not always readily available. However, modelling these equations automatically and in a computationally efficient manner (for optimisation) is a prerequisite for generating task-optimised kinematics.

An important component of the simulation is the application given, as the robot cannot be considered in isolation from the process and other cell components. The simulation itself is based on the virtualised components of the cell (e.g. conveyor belts, grippers), which have the same interfaces as in the real cell, and orchestrates them. Modelling, simulation, and application development therefore ideally take place in the same development environment and in a high-level language. As all information relating to the individual components is also available centrally, this significantly simplifies the creation of the application and reduces communication delays. Manufacturer-specific silos, tools, and programming languages are torn down. In addition to the numerous libraries for application development, physics engines such as PyBullet or MuJoCo are also readily available for process simulation.

2.2 Optimisation

In Section 2.1, it became clear that there are numerous interactions between robot, cell components, and process, which complicate optimisation and require expert knowledge. A good simulation environment offers uniform access and allows for a holistic view. The following paragraphs apply regardless of whether the design is carried out using 'classical' optimization algorithms or via AI agents.

If we consider a given robot structure, we can optimise the DH parameters – in this case we solve a typical continuous optimisation problem, with constraints such as collisions being formulated. There are numerous established methods for this kind of problem (e.g. active set). The task becomes much more complex if the (discrete) number and type of the robot's joints also are determined – this is referred to as a hybrid (discrete-continuous) optimisation problem. Methods exist for this, too, but they are significantly more complex and computationally intensive. Therefore, in practice, one tends to avoid such problems. A good compromise is the automatic selection of application-specific robot structures, for example from a library. Subsequently, geometric parameters for each selected robot, such as segment lengths, can then be optimised. This reduces problem complexity significantly and delivers very good results in a short time.

As we optimise the robot, the trajectory must also be adapted – provided the process allows for this. Typically, two aspects are important here: the path profile (i.e. the accelerations and velocities at which the robot moves) and the path points that define the geometry. This is again a continuous optimisation problem with constraints (such as maximum torque, maximum drive speed, and the position of e.g. via points for the path).

To summarise: selecting from a robot library followed by the adjustment of segment lengths and trajectory is preferable as it reduces computational complexity. Discrete-

continuous optimisation methods are rarely used in practice. If the entire cell is to be optimised, AI agents can also provide support – thanks to standardised interfaces and high-level language programming, they have access to all virtualised components and are thus able to make adjustments to the code and models autonomously and test these automatically (more on this later). Further details on the role of simulation and the use of AI agents for optimisation in an industrial environment can be found in our white paper “AI Agents in Automation”.

2.3 Drives and Control

The mechanical structure of application-specific robots is usually manufactured in small batches or assembled using a construction kit. Thereafter, the drives (motor, gearbox, power electronics) must be integrated mechanically and in terms of control technology. Here, EtherCAT has established itself as a real-time fieldbus standard (in combination with CiA 402). In this respect, the choice of a specific supplier should play a secondary role, provided they adhere to the aforementioned standards.

In Section 2.2, optimised robot models and trajectories are generated within the context of the application. Thanks to modern (simulation) tools, all software modules are available in high-level languages. If these can be seamlessly transferred to the real-world system – without changing the software environment or language – commissioning is greatly sped up. On the contrary, translation into proprietary (robot) languages and path planning algorithms for execution generally results in different geometric and temporal behaviour. Thus, the optimisation results obtained in the simulation are lost. For this reason, traditional control architectures incorporating PLCs (programmable logic controllers) are only of limited suitability. More effective are high-level language- and real-time-capable IPCs (industrial PCs) as a unified control platform with a consistent motion stack – both for application development and for (real-time) control of all components, including robots, via fieldbus.

2.4 Testing, Deployment, and System Operation

Ideally, robot kinematics, path planning, and application development are carried out holistically in simulation and without proprietary restrictions from individual manufacturers. If the real components of the automation cell can be controlled via the same interfaces and without a break in the code base, as described in Section 2.3, the optimisation results can be executed in the cell with just a few clicks immediately.

To ensure reliable operation of the system, the software must (of course) be tested thoroughly beforehand, preferably using automated methods. If this is mainly done during commissioning, troubleshooting becomes costly and time-consuming – which undermines the intended efficiency gains (see also our white paper “Industrial DevOps” on the costs of troubleshooting during various project phases). The automated test pipeline is an integral part of the development environment and builds on the virtualised cell components. It is not only used for the final tests prior to deployment, but continuously as an integral part of the optimisation iterations: the generated results and

the application must be checked each time for collisions, accessibility, correct process flow, etc. Once the designs for the robot and for the other cell components, including the application, have been finalised, tested software modules are readily available. This means there is hardly any additional effort involved in commissioning compared to standard robots – here, too, the application must be tested and the path checked for potential collisions. This advantage would be lost if the code, or parts of it, had to be translated into another language prior to deployment – the test results would consequently lose much of their significance; as it was already the case with the optimisation results from Sections 2.2 and 2.3.

2.5. Conclusion

In order to commercially benefit from the advantages of optimised robots, the solution lies in reducing the efforts for design and software engineering. At the same time, the necessary specialist knowledge (see Sections 2.1 and 2.2) must be kept to a minimum. The Industrial DevOps approach (see Section 3.1) offers ideal prerequisites for this – simulation and automated testing are integral parts of an open, high-level language-based development environment. If the robot is designed using AI agents, this is equally straightforward, as infrastructure, interfaces, and models required for this already exist. We have also seen that robots cannot be considered isolated – the interactions with other cell components and the actual process are so tight that a holistic simulation is required. This, too, is already part of the DevOps philosophy – proprietary tools, which are mostly restricted to individual components, regularly fail in this regard.

The second major step is the commissioning of the robot – as we have already created all necessary software modules in simulation, integrated, and tested them against the (virtual) interface of the drives, all components required are in place. A prerequisite is that the drive supplier adheres to common standards and implements them correctly.

Tested code is also available for the other cell components and the process control. It can be easily uploaded onto a central IPC, which orchestrates the entire system and communicates with the fieldbus components in real time – an additional PLC is no longer required.

3. Case Studies

The following examples were implemented using the open development environment **voraus.pioneer** and the containerised, high-level language-capable control platform **voraus.core**. For a better understanding and for the sake of consistency, we first provide a brief explanation of the underlying philosophy and a description of their working principles (Section 3.1).

Based on Figure 1, Figure 2 illustrates the interaction between voraus.pioneer and voraus.core during optimisation and commissioning. The basic procedure for selecting or optimising task-specific robots is as described in Section 2: given the task, a robot

structure is generated (or selected from a library). The kinematics and motion are calculated using the DH parameters (Section 3.2). This allows for the application being built in simulation and tested automatically. If the performance criteria are met and the tests are passed, the code is deployed to the real system without any adjustments, and the system is immediately ready for operation (Section 3.3 and Section 3.4). If not, the optimisation algorithm or AI makes modifications to the kinematics or selects a different type.

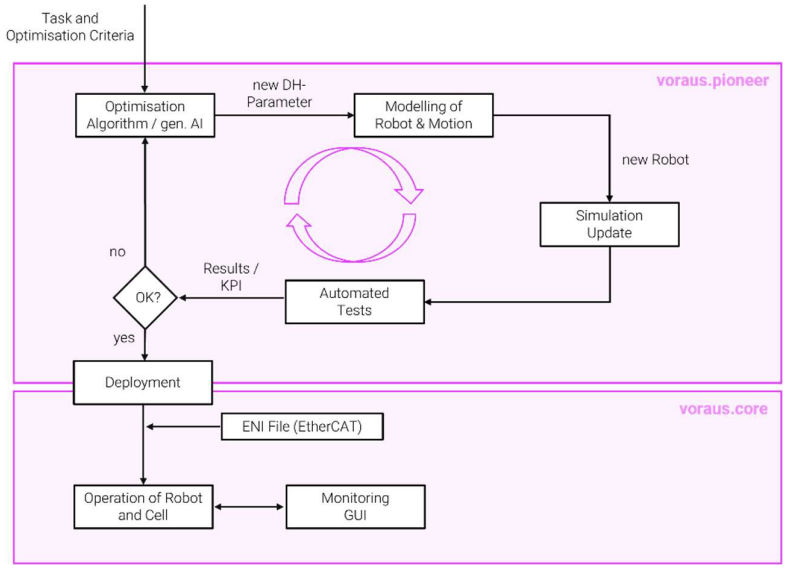


Figure 2: The interaction between voraus.pioneer and voraus.core during optimisation (top) and commissioning/operation (bottom)

3.1 High-Level Language Control and Open Development Environment

The software products voraus.core and voraus.pioneer implement the Industrial DevOps workflow (Figure 3). The aim is to establish efficient development and testing (left-hand side) alongside deployment, operation, and monitoring (right-hand side) – as it is standard practice in IT – within industrial automation (OT). This enables rapid development cycles whilst maintaining high software quality.

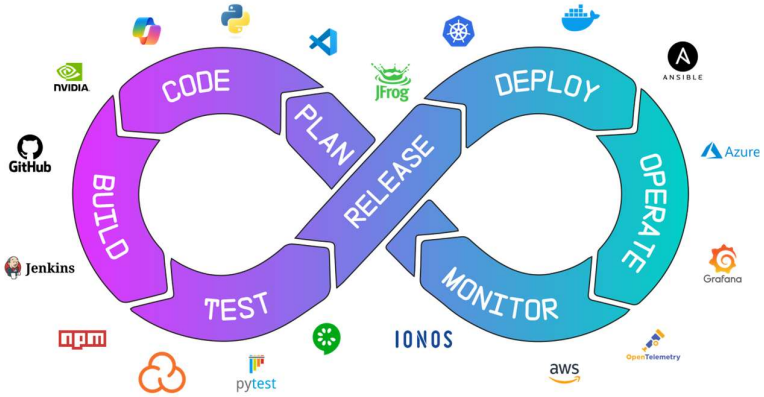


Figure 3: Industrial DevOps workflow at voraus robotik, including an exemplary toolchain (Source of logos: companies' websites)

The prerequisite is a modern, modular, and containerised (and therefore hardware-agnostic) control platform (voraus.core), which connects the individual components of a cell in real time, e.g. via EtherCAT and PROFINET, whilst simultaneously enabling their programming in a high-level language; see Figure 4. Ultimately, the voraus.core forms the basis for the right-hand side of the procedure shown in Figure 3.

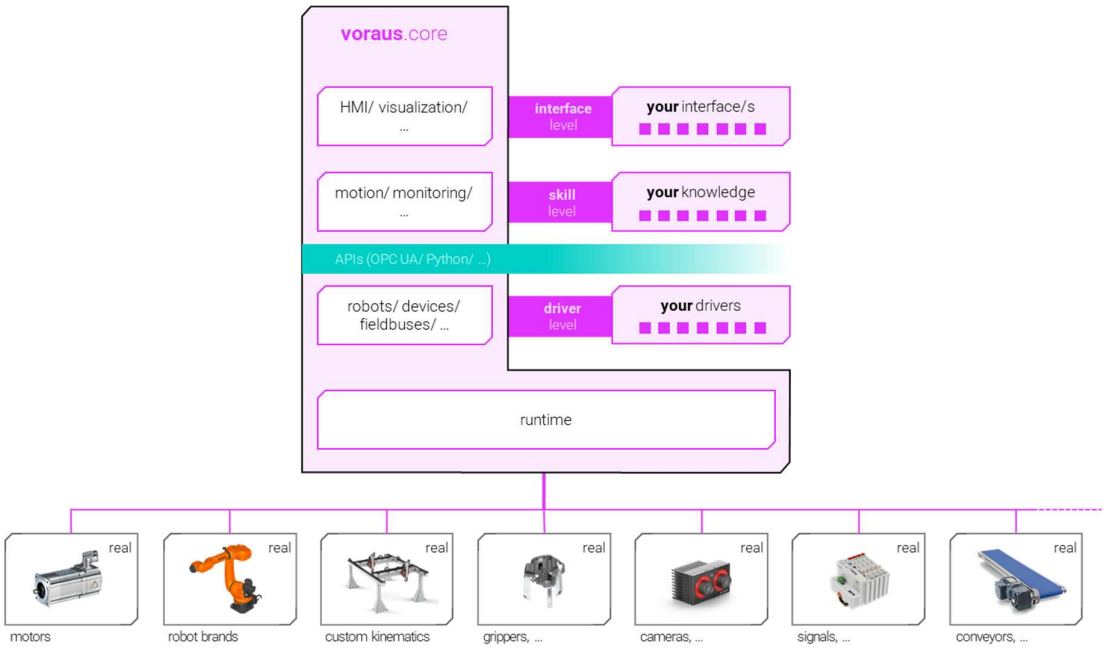


Figure 4: voraus.core – from fieldbus level to high-level language applications

The high-level language capability unlocks entirely new possibilities for OT development – powerful and open (in some cases also open-source) libraries and tools from the IT world can be accessed across all components; vendor lock-in is avoided. A prerequisite for this is the virtualisation of the cell components whilst retaining their interfaces – these are thus available to the developer, even before the actual hardware is in place. Programming, simulation, and automated testing are decoupled from hardware availability. This covers the left-hand side of the DevOps cycle shown in Figure 3 and is implemented by the voraus voraus.pioneer (see Figure 5). Thanks to the open interfaces and programming in Python or Rust, existing libraries for simulation (such as PyBullet, MuJoCo, or NVIDIA Isaac Sim for physical behaviour) or frameworks for testing (e.g. Jenkins) can be efficiently integrated. This also forms the basis for the optimisation and cell design described below. It is important to note that silos are broken down and all components are addressed uniformly. Access to all information and data generated within the system is granted, too. Early and holistic testing in simulation allows errors to be detected early and significantly reduces the risk of system failure.

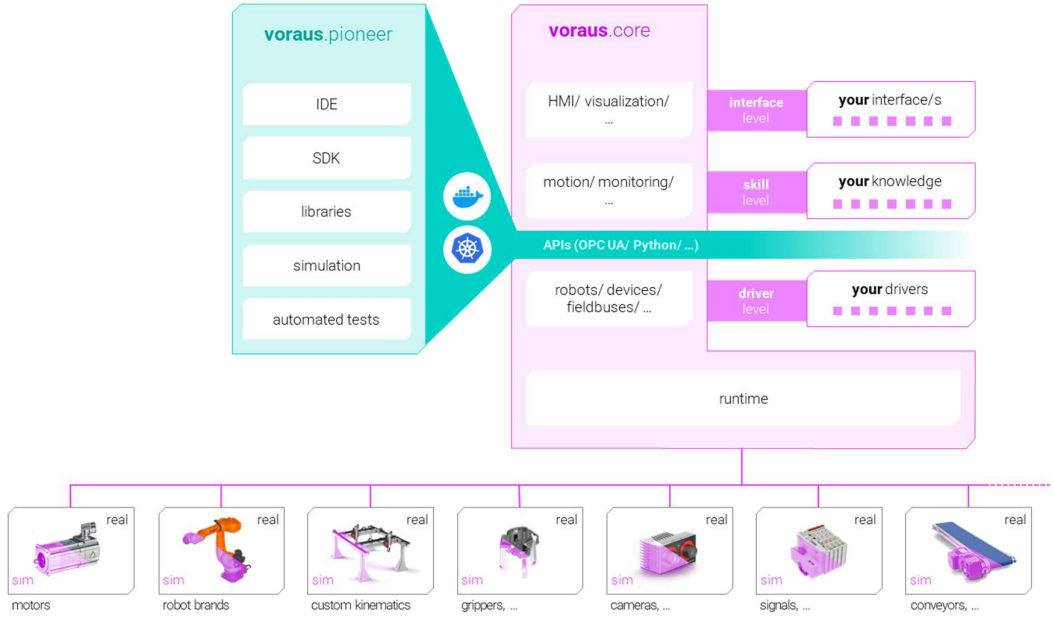


Figure 5: The voraus.pioneer enables holistic development and testing of the entire application in simulation and uses the same interfaces to the components as the real cell

3.2. Simulation and Modelling

As mentioned before, modelling of individual robot kinematics should require as little effort and expert knowledge as possible. In Section 2.1, the DH parameters — which also form the basis for an automated and computationally efficient design process — were briefly presented as an established solution. These parameters can then be passed to an optimisation algorithm or AI as degrees of freedom for adjustment. The basic procedure for kinematics modelling is explained below using a gantry robot as example (Figure 6).

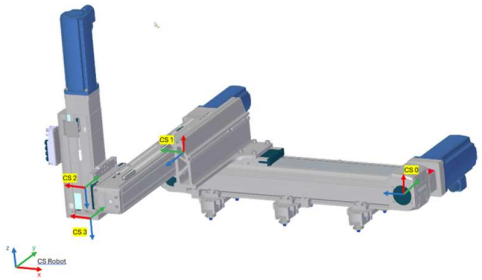
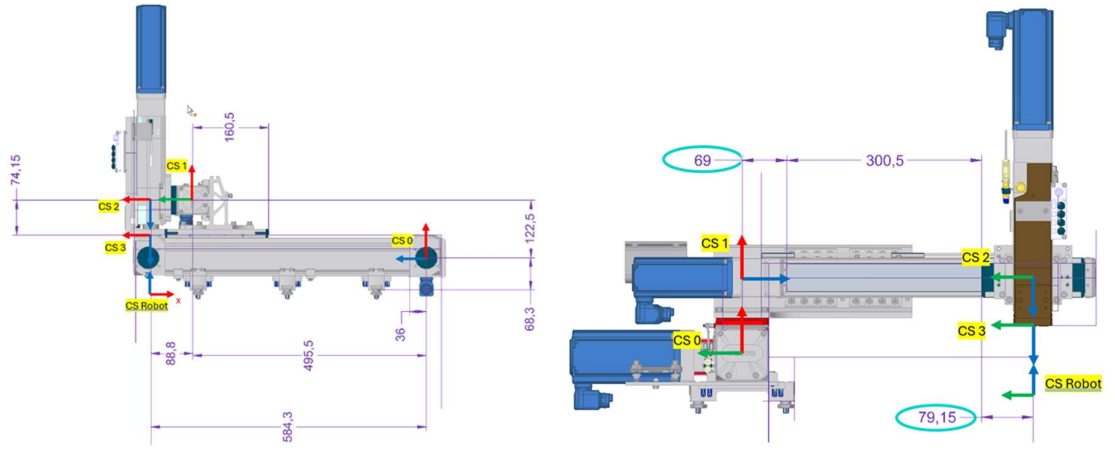


Figure 6: Considered gantry kinematics with three degrees of freedom (*x*-, *y*- and *z*-direction)

The DH parameters can be derived directly from the robot model and are given in Figure 7.

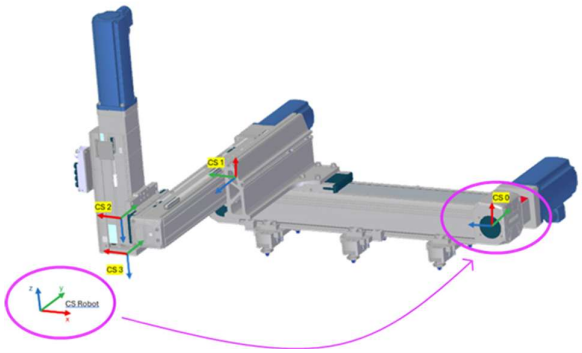


Joint	θ	d [mm]	a [mm]	α [rad]
1	0	q1 + 196.5	122.5	π/2
2	π/2	q2 + 148.15	88.8	-π/2
3	0	q3 + 74.15	0	0

Joint	θ	d [mm]	a [mm]	α [rad]
1	0	q1 + 196.5	122.5	π/2
2	π/2	q2 + 148.15	88.8	-π/2
3	0	q3 + 74.15	0	0

Figure 7: DH parameters (the definition of the coordinate systems is omitted here)

Finally, the base coordinate system needs to be provided (see Figure 8). It defines the pose of the robot base relative to the cell.



x [mm]	y [mm]	z [mm]	Rx [rad]	Ry' [rad]	Rz'' [rad]
584.3	448.65	68.3	0	-π/2	0

Figure 8: Definition of the base coordinate system

These parameters are transferred to a JSON file (see Figure 9). Thereafter, both voraus.core and voraus.pioneer are able to automatically calculate the robot's kinematics, including direct and inverse kinematics. From this point onwards, no further expert knowledge is required. The algorithms are identical in both the development environment and the controller – thus resulting in the consistent behaviour as required above. Direct and inverse kinematics are also used in the motion stack, which provides, amongst other things, path planning, blending, and various types of stop. This allows for analysis of workspace and reachability as well as determination of cycle times. If the robot structure is enriched with simple envelopes that scale with the kinematic parameters (e.g. segment length), collisions with other interfering contours can already be detected in simulation. This forms the basis for a fully automatic design.

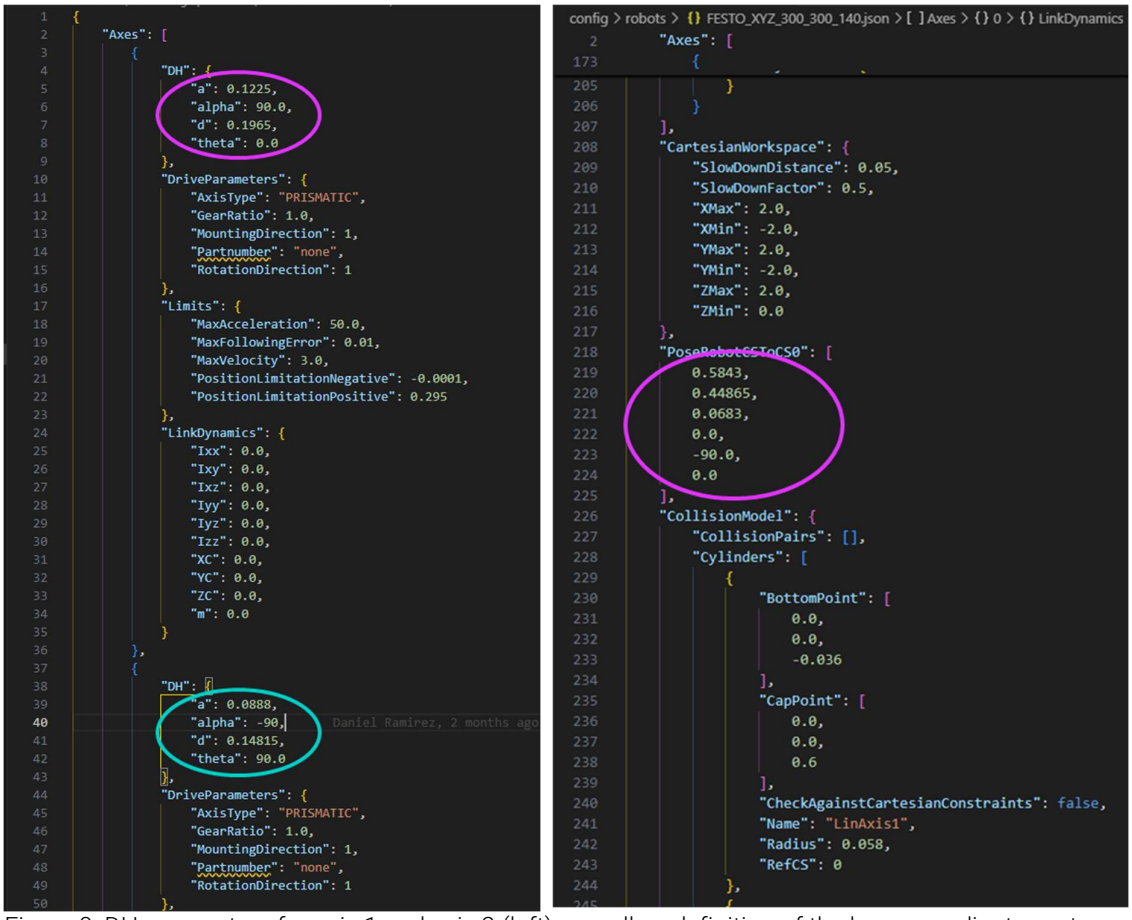


Figure 9: DH parameters for axis 1 and axis 2 (left) as well as definition of the base coordinate system (right) in the robot configuration file

In addition, it is possible – although this is more relevant to the actual operation of the robot in the cell – to specify inertia and friction parameters, which are then used in the robot's dynamics and control (e.g. feedforward control for improved path accuracy).

3.3 Optimisation Using a Robot Library

The modelling approach described in Section 3.2 still requires expert knowledge (albeit to a limited extent), despite the simplifications. An obvious next step is the usage of a library of existing robots that can be integrated fully automatically into the simulation. Individual customisation is no longer required; instead, the optimal selection of a specific type is made. As the exact contours and dynamic parameters are typically available, the simulation gains additional value. An example is the combination of voraus.pioneer with the *autonox Finder* from autonox Robotics or the *robot configurator* from igus.

In the following, robots from autonox Robotics are integrated into a pick-and-place application. The first step involves creating a simulation of the cell with its relevant components and programming the application in Python in the voraus.pioneer (Figure 10). The robot shown on the right is an example of a selection made from the product portfolio.

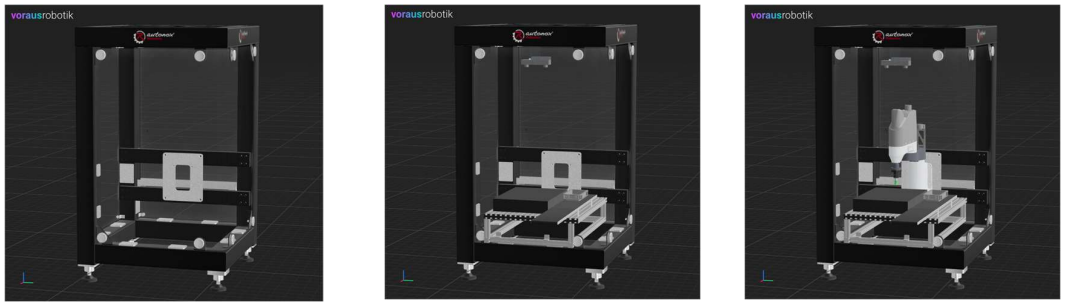


Figure 10: Step-by-step setup of a pick-and-place application in simulation with an autonox Robotics robot, including boxes and conveyor belt

The optimisation objective for pick-and-place is typically the minimisation of the cycle time for one (or more) pick(s). At the same time, the path must be collision-free and the conveyor belt and boxes (containing the objects to be picked) must be within the robot's workspace.

When the optimisation is started, the algorithm selects a robot from the library, updates the simulation and path planning, and determines the cycle time. In a second step, the operator selects the preferred configuration. In this way, cell size/number of boxes and robot can be optimized simultaneously, see Figure 11. Here, the user can make a final choice between the time-optimised variant (right) or taking into account a potential future increase in the number of boxes (left), thereby accepting a slightly longer cycle time and larger cell layout. Ultimately, this makes investment decisions more transparent and the return on investment easier to calculate.

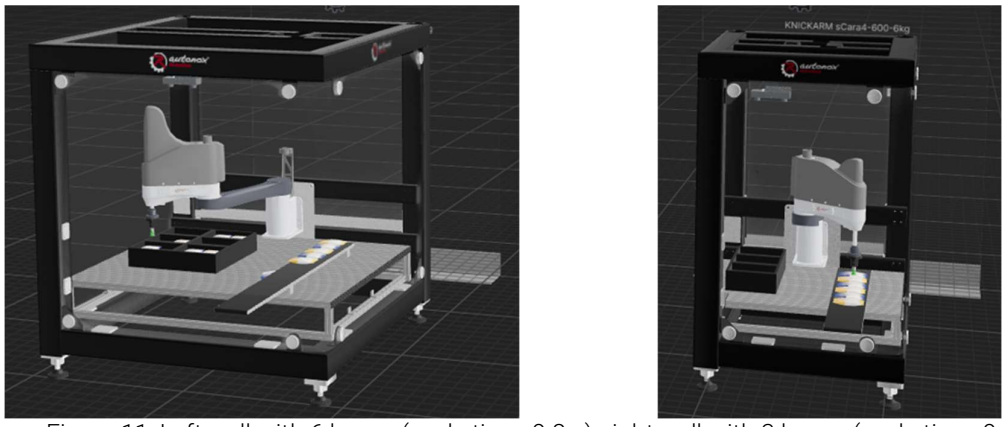


Figure 11: Left: cell with 6 boxes (cycle time: 9.8 s); right: cell with 3 boxes (cycle time: 9.4 s)

Of course, robots from different manufacturers can also be compared – the voraus robotik software stack is robot-agnostic from an application perspective (see Figure 12 – showing examples for autonox, Fanuc, and KUKA). In simulation, cycle time, reachability, collisions, etc. are determined for the respective robot type or manufacturer, and the best solution is selected.

Furthermore, application and parameterisation are strictly separated. Figure 12 illustrates this by a palletising application including a conveyor belt: robots from different manufacturers perform the same task in different layouts. Each cell can be

automatically optimised and tested based on individual criteria: e.g. robot selection and motion with regard to cycle times, including collision checks, or the positioning of individual components to minimise space requirements. For example, in the case of a simple pick-and-place cell, cycle times can be reduced from 3.4 s to 2.5 s by simply selecting the best suited robot – a reduction of more than 25% at the touch of a button, resulting in a huge efficiency gain. Once the final configuration has been selected, the real cell is built (Figure 12, bottom left) and the control program (robot, conveyor belt, camera integration, etc.) is deployed within minutes.

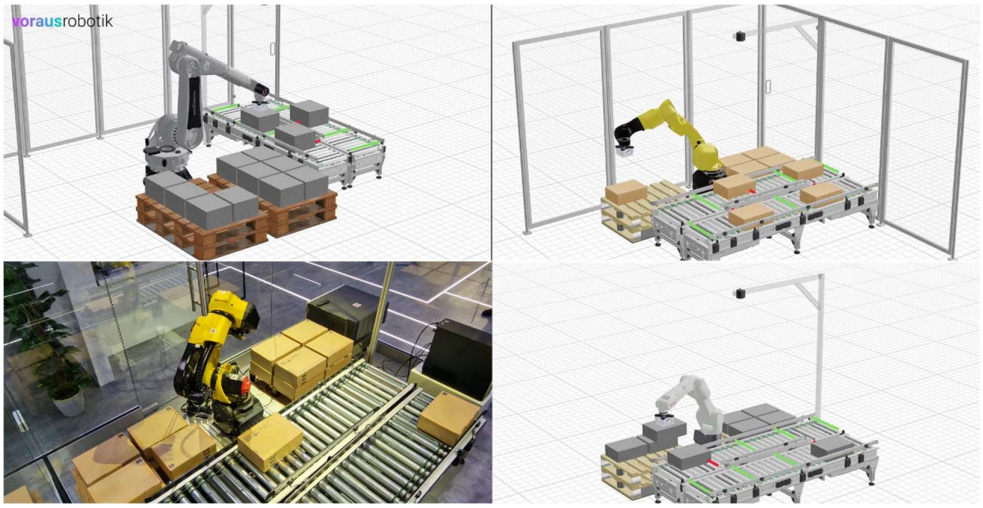


Figure 12: Robots from different manufacturers can be compared fully automatically in simulation across various cell layouts, thereby guaranteeing optimal set-up

In case of the autonox robots, once the optimal mechanics has been defined, drives and control systems still need to be integrated – this is subject of Section 3.4. Both the application and the robot control software can be used directly for the real cell – no additional effort is required during deployment.

3.4 Drives and Control

As mentioned above, a major advantage of the outlined approach is the centralized control of all components using standard high-level languages and avoiding (proprietary) silos. Consequently, no PLC is used for real-time communication (fieldbus) neither, but rather a cost-effective IPC. The containerised voraus.core runs on the IPC (or edge device) and includes software stacks for EtherCAT and PROFINET fieldbus protocols.

Below, we examine the drive integration via EtherCAT: the first step involves generating the ENI (EtherCAT Network Information) file, which describes, amongst other things, the specific network topology. Furthermore, if the drives are CiA 402-compliant, no further integration effort is required. Together with the robot parameters, the ENI file is transferred to the voraus.core, and access to the robot's drives via high-level language is immediately possible (see Figure 13). The voraus motion stack ensures synchronised

multi-axis motion, and the programmer can concentrate on writing the application, with centralised and uniform access to all components.

For example, a dashboard can be quickly connected via OPC UA, whereas its creation is possible using open-source tools (e.g. Grafana, pydash, rerun.io) in almost no time. As the voraus.core supports IT standards and interfaces, integration into higher levels of the control pyramid (e.g. ERP, MES) is also straightforward.

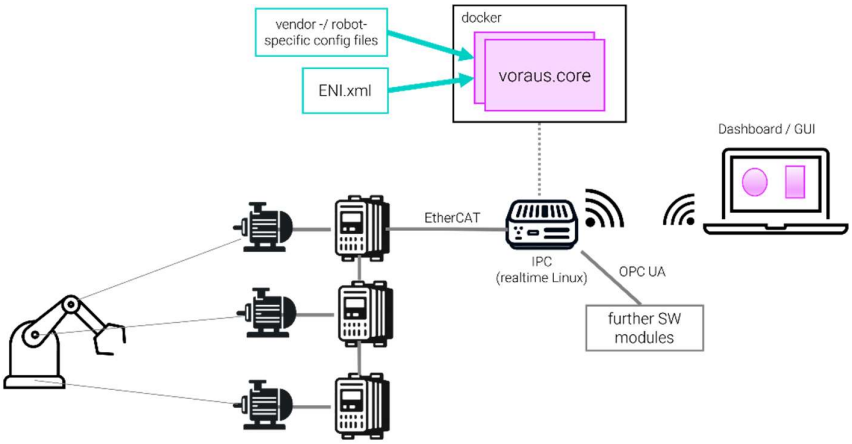


Figure 13: Integration of the robot drives (ENI file), the robot kinematics, and other cell components (including dashboard / GUI)

Finally, it should be noted that uniform access to all data, the central control architecture, and the consistent use of high-level languages for all components are essential prerequisites for data analysis (e.g. anomaly detection) and for the deployment of AI agents. Further details can be found in the white paper “AI Agents in Automation”.

3.5 Testing and Deployment

Thanks to the abstraction and virtualisation of all cell components, the development can start regardless of the specific choice and availability of the actual hardware. The application itself, and in particular the handling of errors and edge cases, can be automatically tested in the virtual (simulation) environment. This is an integral part of the DevOps philosophy and is, of course, also applied when optimising or selecting robot kinematics – without any additional effort. The test pipeline covers all steps involved in creating the release, including static code analysis, unit, motion, and Docker tests, as well as building of the software artefacts. Application-specific analyses such as collisions and KPIs (e.g. cycle time) can also be easily integrated. The tests are then run automatically at each optimisation step to ensure high-quality code to be delivered. This reduces the time required for an update from days or weeks to just a few minutes, whilst significantly lowering the risk.

The result of the gantry kinematics discussed in Section 3.2 is shown in Figure 14. The code is tested in the simulation and can be deployed to the real cell with just a few clicks. Furthermore, no breaks in the toolchain occur, as only one central control computer needs to be updated.

Figure 14: Individual kinematics including automatically tested control code – real plant and simulation

4. Conclusion and Outlook

Designing individual, task-optimised robots or selecting them automatically from a catalogue – end-to-end simulation is the solution for efficiency. This allows for the evaluation of the effects of an adjusted workspace or path on the process and other cell components. If simulation, application, and real time control of the cell are programmed in the same high-level language, deployment is highly efficient. However, this requires comprehensive and automated validation of the software modules already during the development phase. The aforementioned aspects are an integral part of the Industrial DevOps workflow, so no additional effort is required in this regard. Design and deployment of custom robots become highly cost-effective.

As described above, in addition to optimising the robot, the cell or system can also be included in the design process. Given the complexity of the resulting optimisation problem, this is hardly possible without extensive expert knowledge. Here autonomous AI agents play a key role: if they are given access to an initial design of a non-optimised cell and to the relevant data, they can formulate hypotheses, test them in simulation, and modify them if necessary. This results in a cell that has been optimised, for example, in terms of operating costs or cycle time.

5. Company Information and Contact Details

Further information about our software platform and how we can support you in industrial automation can be found on our website www.vorausrobotik.com. Our documentation, featuring numerous examples, is available here: https://docs.vorausrobotik.com/.

Contact Details

voraus robotik GmbH | Carl-Buderus-Str. 7 | 30455 Hanover | Germany
vorausrobotik.com

Contact

Tobias Ortmaier | tobias.ortmaier@vorausrobotik.com