KI-Agenten in der Automatisierung

Wie groß ist der Nutzen von KI in der industriellen Automation heute schon und was sind die Voraussetzungen?

vorausrobotik

Carl-Buderus-Str. 7 | 30455 Hannover | **voraus**robotik.com

Kurzzusammenfassung Dieses White Paper zeigt, wie generative KI und insbesondere KI-Agenten jenseits des Hypes schon jetzt einen realen wirtschaftlichen Nutzen in der industriellen Automation liefern. Hierzu wird eine Pick-&-Place Anwendung betrachtet – dabei konnte die Pickrate (erfolgreiches Greifen von Tüten) auf über 99 % und die Pickgeschwindigkeit um mehr als 16 % gesteigert werden. Damit einher geht eine deutliche Zeitersparnis in der Software-Entwicklung und dem Deployment auf die reale Anlage um bis zu 80 %. Neben den hierfür benötigten Voraussetzungen für den Einsatz im Produktivbetrieb werden auch die aktuellen Grenzen generativer KI diskutiert.

1. Einleitung und Motivation

Im White Paper „Industrial DevOps“ haben wir die Bedeutung der modernen SW-Entwicklung in der industriellen Automatisierung betrachtet. Die einheitliche Programmierung aller Komponenten einer Automatisierungsanlage in Hochsprache (z. B. Python oder Rust) und die Virtualisierung ebendieser für Simulation, Test und Optimierung ermöglichen schnelle Release-Zyklen in hoher Qualität für OT-Anwendungen. Darauf aufbauend betrachtet dieses White Paper den Einsatz von KI zur Optimierung industrieller Automatisierungslösungen, wobei auch hier die Code-Qualität im Vordergrund steht. Eine hohe Testabdeckung ist essenziell, um Anlagensteuerungen ohne Risiko einer versehentlichen Beschädigung zu aktualisieren und mit hoher Verfügbarkeit im Feld zu betreiben (Industrial Grade). An einem Pick-&-Place Beispiel betrachten wir Ansätze, um den Output der Zelle zu steigern. Im Zentrum steht dabei nicht die Frage nach dem bestmöglichen KI-Tool – diese sind ohnehin einem rasanten Wandel unterworfen – sondern vielmehr wie diese modernen Werkzeuge zur Effizienzsteigerung in den Industrial DevOps-Workflow eingebunden werden und welche Voraussetzungen für deren industriellen Einsatz gegeben sein müssen.

Die klassische Programmierung industrieller Automatisierungslösungen stößt zunehmend an Kapazitätsgrenzen. Der hohen Nachfrage nach Experten, die über spezielle Programmier- (strukturierter Text, KUKA KRL, FANUC KAREL, …) und proprietäre Tool-Kenntnisse verfügen, bedingt durch einen zunehmenden Automatisierungsgrad sowie steigende Komplexität von Produktionsanlagen und

Prozessen, steht deren abnehmende Verfügbarkeit aufgrund des demographischen Wandels gegenüber. Daraus resultieren hohe Kosten und lange Projektlaufzeiten sowie Druck auf das Management (vornehmlich CTO und CDO), Maßnahmen zur Sicherstellung der Wettbewerbsfähigkeit zu identifizieren und umzusetzen.

Ein Schlüssel ist der Einsatz generativer KI im Industrial DevOps-Kontext. Dies können Tools zur Unterstützung bei der Fehlersuche im Feld, zum Anforderungsmanagement oder zur automatischen Dokumentation sein. Besonderes Potential weisen insbesondere KI-gestützte Werkzeuge auf, die bestehende Lösungen optimieren (also bei bereits installierten Anlagen zu einer Effizienzsteigerung führen) oder Software-Code automatisch generieren und damit die Entwicklungsdauer drastisch verkürzen können. Beiden Ansätzen ist gemein, dass der Entstehungsprozess für den Anwender nicht immer transparent ist und diese Software schlussendlich auf teuren Anlagen Prozesse steuert. Fehler oder Lücken im Code können daher zu Beschädigungen und Stillstand führen und die Effizienzgewinne ins Gegenteil verkehren. Insofern ist eine hohe Testabdeckung von besonderer Bedeutung, wobei diese durch zahlreiche Corner- bzw. Edge-Cases erschwert wird. In diesem White Paper betrachten wir Beispiele zur Effizienzsteigerung sowie die erforderlichen Rahmenbedingungen, um automatisch optimierten oder generierten Code in industriellen Anlagen zielführend einzusetzen. Als Software-Werkzeug kommen hierbei vorzugsweise generative KI-Tools (sog. große Sprachmodelle) als integraler Bestandteil von KI-Agenten zur Anwendung, sodass wir zunächst einen Blick auf deren Funktionsweise werfen wollen.

2. Generative KI für die industrielle Automatisierung

Dieser Abschnitt skizziert kurz die grundlegende Funktionsweise von Large Language Models (LLMs) und die damit verbundenen Herausforderungen.

2.1 Wie generative KI funktioniert

In den hier genannten Szenarien werden bevorzugt LLMs als Sonderform der tiefen neuronalen Netze eingesetzt. Dabei wird der Eingang (Prompt) in Teilfragmente (Tokens) zerlegt und anschließend darauf aufbauend wahrscheinlichkeitsbasiert eine Vorhersage für den Ausgang berechnet. Dieser kann natürlichsprachlich sein, aber auch Formeln oder Software sind möglich. Für das Training des neuronalen Netzes und damit der Wahrscheinlichkeiten werden große Datenmengen benötigt. Hierfür bedienen sich die Anbieter im Wesentlichen dem öffentlich zugänglichen Material im Internet.

Eine Sonderform sind sog. RAG (retrieval augmented generation) Modelle. Hier wird das LLM um in der Firma vorliegende proprietäre (lokale) Informationen angereichert, um spezifischere Antworten zu liefern. Dies reduziert auch die Wahrscheinlichkeit von sog. Halluzinationen (siehe Abschnitt 2.2).

2.2 Die besonderen Herausforderungen

Sicherheit und Zuverlässigkeit

Der Rückgriff auf das Internet zum Training der LLMs ist auch gleichzeitig eine der größten Schwächen: Die Güte der Ausgabe ist stark von der Qualität und Quantität der Trainingsdaten abhängig. Stehen nur begrenzt Daten zur Verfügung oder sind diese schlecht (z. B. schlechter Programmierstil, Fehler in Formeln), so lernt das LLM auf diesen Informationen und erzeugt im Zweifelsfalle ein schlechtes / unbrauchbares Ergebnis. Auch verfügen die LLMs über kein Verständnis des für OT-Anwendungen so wichtigen zeitlichen Verhaltens. Ebenso ist das allseits bekannte Phänomen der Halluzinationen, also dem Erzeugen von Inhalten, die sich nicht in den Trainingsdaten wiederfinden, trotzdem aber korrekt erscheinen, ein gegenwärtiges Problem. Wird so generierter Code ohne ausführliches Testen auf die reale Anlage aufgespielt, können Beschädigung und Stillstand die Folgen sein. Dies schränkt den industriellen Einsatz aktuell stark ein – ein ggf. fehlerhaftes Dashboard mag vielleicht noch akzeptabel erscheinen, eine fehlerhafte Maschinensteuerung definitiv nicht.

Auch besteht die Gefahr, dass der Code (ggf. bereits bekannte oder behobene) Sicherheits-Schwachstellen beinhaltet und die Anlage damit anfällig für Cyber-Kriminalität wird. Hintergrund ist, dass das Training sehr kosten- und zeitintensiv ist und nur Informationen bis zu einem bestimmten Stichtag berücksichtigt – Neuerungen finden daher ggf. keinen Eingang. Dies stellt gerade im Kontext des CRAs (Cyber Resilience Act) eine große Herausforderung dar. Theoretisch wäre sogar das gezielte Einschleusen von Schadcode denkbar.

Integration in bestehende IT/OT-Landschaften

Typischerweise laufen industrielle Applikationen nicht zentral auf einem Rechner, sondern verteilt über mehrere Steuerungs-Komponenten: Speicherprogrammierbare Steuerungen (SPS) koordinieren bspw. Förderbänder, eine Robotersteuerung die Roboterbewegung und eine Kopfsteuerung orchestriert die Zelle und kommuniziert mit einer intelligenten Kamera. Die Herausforderung besteht in den dabei involvierten proprietären Programmiersprachen (z. B. strukturierter Text und roboterspezifische Sprachen) sowie der individuellen Rechner- bzw. Kommunikationstopologie. Hier muss sich der automatisch generierte Code als Teillösung einfügen und dabei ggf. noch weitere herstellerspezifische Besonderheiten (z. B. Siemens vs. Rockwell vs. …) berücksichtigen. Zusätzlich ist, siehe oben, ausreichend Trainingsmaterial erforderlich. Aus Sicht der KI wäre eine zentrale und einheitliche Steuerung aller Komponenten (weniger Kontextwissen erforderlich) in Hochsprachen (für die entsprechend mehr Trainingsdaten existieren) die vielversprechendere Wahl.

Rechtliche Aspekte und geistiges Eigentum

Das von der KI generierte Ergebnis unterliegt nach gängiger (deutscher) Rechtsauffassung nicht dem Urheberrecht, da dieses eine persönliche Schöpfung

voraussetzt. Wird jedoch bspw. Code repliziert, der in den Trainingsdaten vorhanden ist, kann eine Urheberrechtsverletzung begründet sein, falls die entsprechenden Trainingsdaten selbst urheberrechtlich geschützt sind (Quelle: https://kpmg-law.de/ki-und-urheberrecht-was-ist-beim-einsatz-von-llms-erlaubt/). RAG-Architekturen, die auf eine interne Wissensbasis zurückgreifen, haben das Potential, das Risiko von Urheberrechtsverletzungen zu reduzieren, auch wenn sie nicht gänzlich ausgeschlossen werden können. Insofern bleibt dieser Aspekt mit Unsicherheiten behaftet.

Während die urheberrechtlichen Aspekte schwer zu fassen und z. T. noch letztinstanzlich zu klären sind, sind die Haftungsfragen bei Fehlfunktionen deutlich klarer geregelt: Die Neufassung der Produkthaftungsrichtlinie im Jahre 2024 weitet die Haftung von SW-Herstellern deutlich aus. Auch ein Verstoß gegen den Cyber Resilience Act (CRA) kann eine Haftung begründen (Quelle: https://www.heise.de/hintergrund/Software-Anbieter-aufgepasst-Sie-haften-jetzt-fuer-fehlerhafte-Produkte-10027145.html). Insofern kommt dem automatisierten Testen der Software eine essenzielle Rolle zu; manuelles Testen ist zu zeitaufwändig und kostenintensiv und erlaubt daher keine schnellen Update-Zyklen.

3. Simulation und Hochsprache als Schlüssel

Was lässt sich nun aus den in Abschnitt 2 genannten Besonderheiten ableiten?

Essenziell ist, dass viele Daten zum Training der LLM vorliegen, da letztendlich Wahrscheinlichkeiten optimiert werden. Diese Bedingung lässt sich leichter erfüllen, wenn die Automatisierung in gängigen Hochsprachen programmiert wird, die eine deutlich größere Verbreitung im Netz als bspw. strukturierter Text, KRL etc. aufweisen. Dies hilft auch, das in Abschnitt 2.2 erwähnte Problem von fehlerhaften Code-Beispielen beim Training von LLMs zu egalisieren – aufgrund der großen Menge an Trainingsdaten in Hochsprache fallen diese statistisch deutlich weniger ins Gewicht und die Qualität der Netze wird so gesteigert. Ebenso vereinfacht ein zentraler, echtzeitfähiger Steuerungsrechner für die gesamte Zelle den Prozess der Code-Erzeugung, da die individuelle (Rechner-/Software-/Kommunikations-)Architektur der Zelle nicht berücksichtig werden muss. Eine hierfür geeignete Lösung ist die Steuerungsplattform **voraus.core** (siehe Bild 1). Eine moderne Software-Architektur erlaubt das Ansprechen aller Komponenten einer Automatisierungszelle über standardisierte Schnittstellen (EtherCAT, PROFINET, OPC UA etc.) in Hochsprache und in Echtzeit.

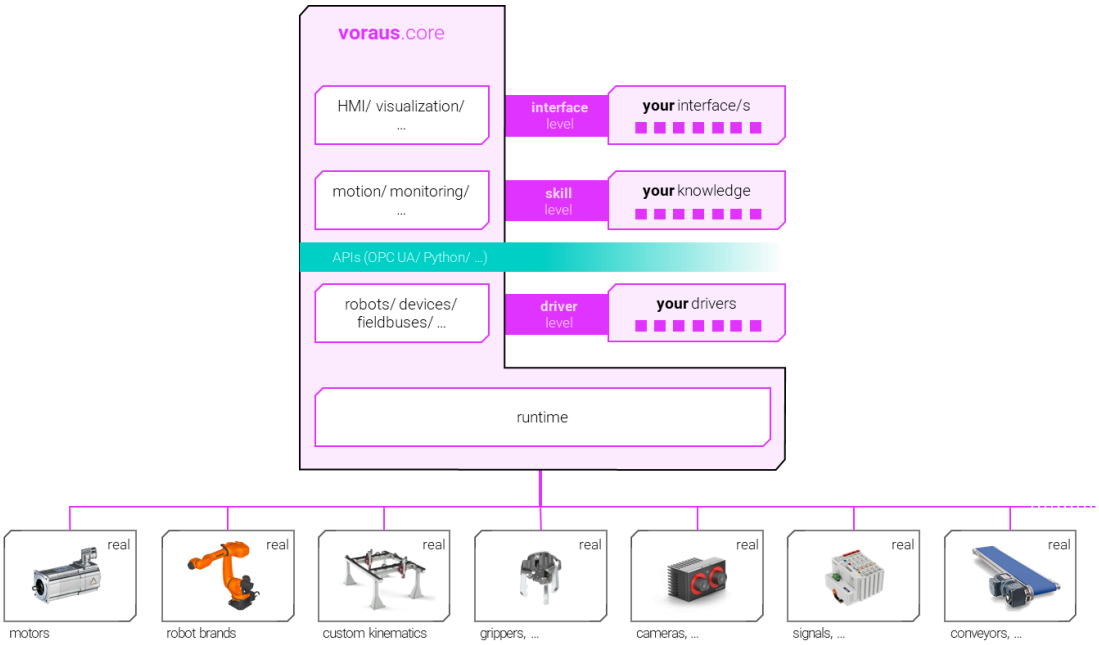

Bild 1: Steuerungsplattform voraus.core: moderne Architektur und offene Schnittstellen ermöglichen die direkte Ansprache von Feldbuskomponenten aus der Hochsprache in Echtzeit

Offen bleibt jedoch zunächst die Frage nach Qualität und Vollständigkeit des optimierten bzw. generierten Codes. Hier spielen insbesondere die zahlreichen Edge-Cases sowie Fehler- und Sonderfälle eine große Rolle, die eine hohe Testabdeckung erschweren, für die Vermeidung von Stillständen oder gar Maschinenschäden (und dem damit einhergehenden Haftungsrisiko) jedoch essenziell sind. Unstrittig ist also, dass der erzeugte Code ausführlich zu testen ist, bevor er auf die reale Anlage aufgespielt wird. Eine Simulationsumgebung, die die einzelnen Komponenten virtualisiert zur Verfügung stellt und in derselben Programmiersprache ohne Brüche wie die reale Zelle programmiert wird, ist der Schlüssel. Nur so kann die Automatisierungslösung in ihrer Gesamtheit (und nicht wie oben beschrieben nur bruchstückhaft) getestet und eine hohe Testabdeckung in einem frühen Stadium des Entwicklungsprozesses erreicht werden. Wichtig ist, dass dabei dieselbe Code-Basis wie in der realen Zelle zur Anwendung kommt. Eine Übersetzung in herstellerspezifische Programmiersprachen vor dem Deployment reduziert die Aussagekraft der Testergebnisse drastisch. Die Entwicklungsumgebung **voraus.pioneer** (siehe Bild 2) setzt diesen Ansatz um, und stellt eine einheitliche Code-Basis zwischen Entwicklung, Simulation sowie realer Zelle sicher.

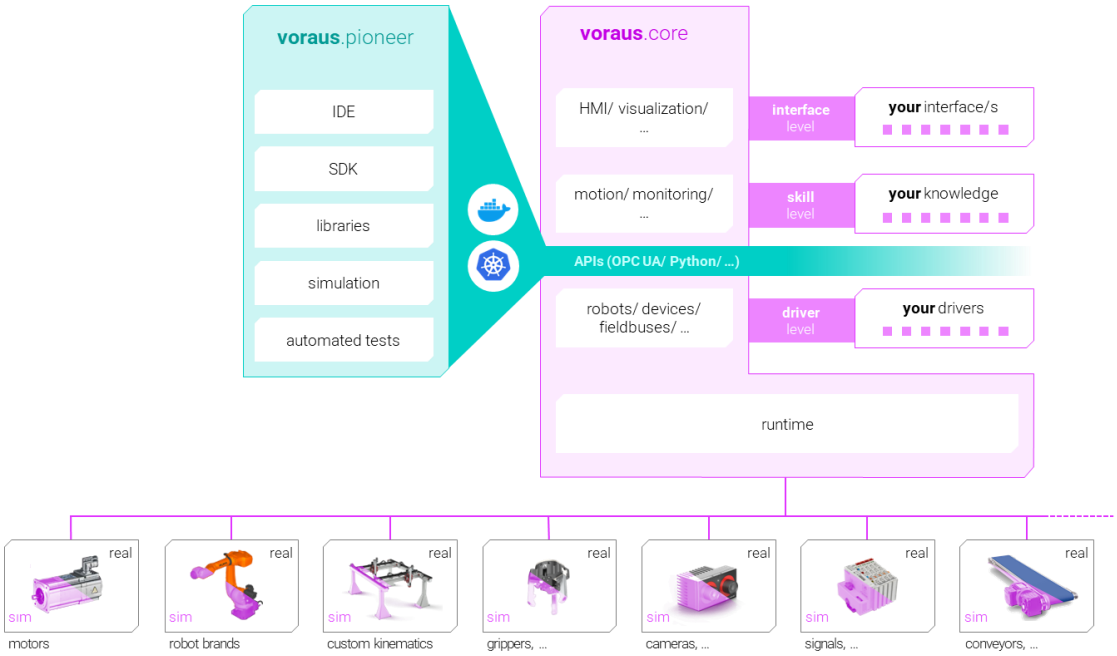

Bild 2: Der voraus.pioneer erlaubt das holistische Testen der gesamten Applikation in der Simulation und verwendet dieselben Schnittstellen zu den Komponenten wie die reale Anlage

Sind diese Voraussetzungen gegeben, so können KI-Funktionalitäten nahtlos in den Industrial DevOps-Workflow eingebunden werden (siehe Bild 3 am Beispiel des voraus.pioneer). Offene und auf IT-Standards basierende Schnittstellen erlauben eine flexible und schelle Integration verschiedenster Tools und Bibliotheken. Die hohe automatisierte Testabdeckung entdeckt Fehler zu einem frühen Zeitpunkt und reduziert das Risiko bei der Inbetriebnahme. Wie in Abschnitt 4 gezeigt, eröffnen sich völlig neue Ansätze zur Optimierung wichtiger KPIs: So können KI-Agenten Verbesserungen vorschlagen, die dann (ebenso wie der übrige Code) der Test-Pipeline unterzogen und geprüft werden. Dadurch ist es möglich, bspw. Taktzeiten oder Platzbedarfe einfach und zu einem frühen Zeitpunkt automatisiert zu optimieren und Wettbewerbsvorteile auf Knopfdruck zu sichern. Letztendlich handelt es sich hierbei um eine industrielle Umsetzung einer Agentic Feedback Loop (AFL) – also dem automatisierten Generieren, Testen und Korrigieren von Theorien durch KI-Agenten. Wie in Bild 3 ersichtlich, müssen sich KI-Agenten nahtlos in die Architektur der Steuerungsplattform einfügen.

Schlussendlich bleibt noch die Frage, wie die (zahlreichen) Tests erstellt werden. Testspezifikation und -implementierung sind inhärenter (wenn auch zumeist unbeliebter) Bestandteil der SW-Entwicklung. Neben der zur Zeit häufig noch üblichen manuellen Spezifikation und Implementierung existieren auch hier Ansätze Tests bspw. aus dem Lastenheft und aus den im Rahmen der Spezifikations-/Entwurfsphase zu erstellenden Dokumenten (linke Seite im V-Modell) automatisch zu generieren. Besonders vielversprechend sind dabei unter anderem Unit Tests, da die Funktionalität klar beschrieben ist und in den Trainingsdaten hierfür häufig zahlreiche Beispiele existieren. Oft werden die automatisiert erstellten Tests mit durch den Programmierer vorgegebenen kombiniert – so auch in unserem Fallbeispiel in Abschnitt 4.2. Wie im White Paper Industrial DevOps gezeigt, senkt die frühzeitige Entdeckung von Fehlern

deren Kosten für die Behebung signifikant: Werden beispielsweise Fehler erst während der Inbetriebnahme entdeckt, so sind die Kosten für die Behebung um den Faktor 40 höher als wenn dies während der Implementierungsphase geschieht (Quelle: Jones, Caspers. Applied Software Measurement: Global Analysis of Productivity and Quality). Dies sollte (neben der eigentlichen Produkthaftung) ausreichend Motivation für ein gutes automatisiertes Testmanagement (bspw. mittels Jenkins) und eine hohe Testabdeckung sein.

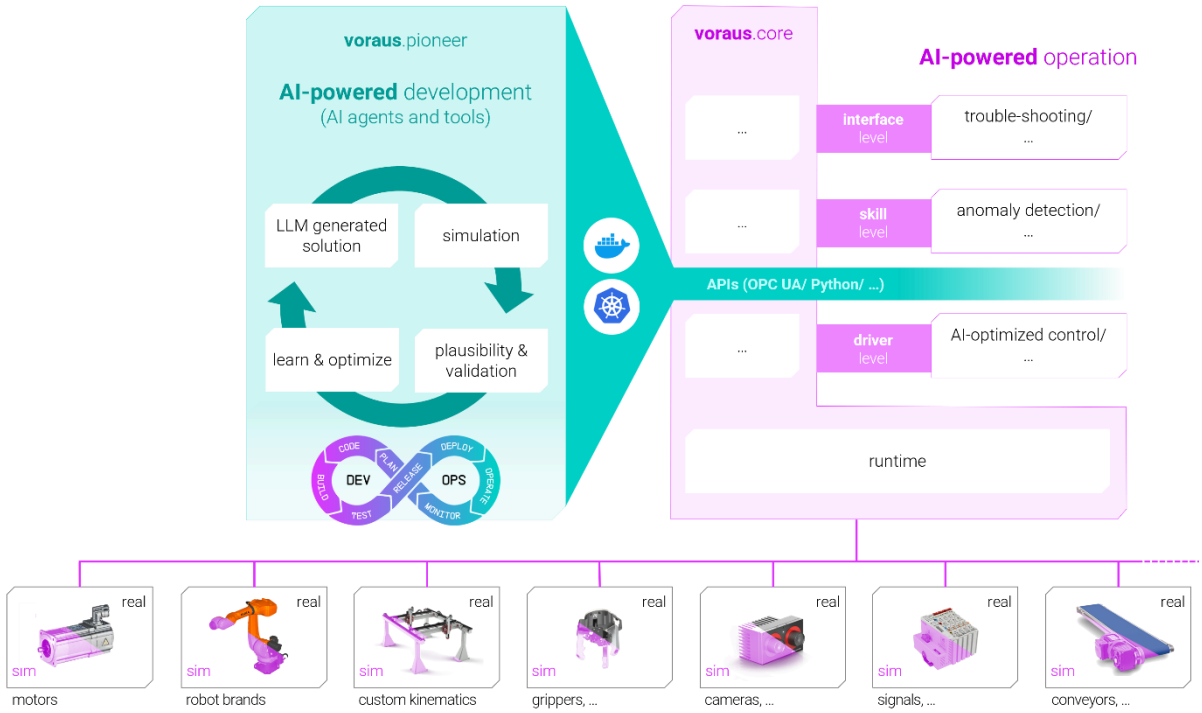

Bild 3: Der voraus.pioneer ermöglicht einen KI-getriebenen SW-Entwicklungsprozess

4. Fallstudien

Die in Bild 4 gezeigte Pick-&-Place-Zelle, bestehend aus einem Roboter mit Sauggreifer, einer Roboception Kamera, einem Förderband zur Ausgabe, drei Kisten mit verschiedenen Tüten sowie einem Touch-Display, um Aufträge zu platzieren, wird vom voraus.core zentral gesteuert. Die gesamte Applikation inkl. EtherCAT-Kommunikation läuft zentral auf einem kleinen IPC; eine SPS kommt nicht zum Einsatz. Die Programmierung aller Komponenten und der Anwendung erfolgt in Hochsprache.

Bild 4: Pick-&-Place-Zelle mit Roboter, Förderband, drei Kisten und Sauggreifer

Der vereinfachte Ablauf für die Abarbeitung eines Auftrags ist in Bild 5 gezeigt: In einer Scanpose, die die Sicht der Kamera auf die Boxen nicht verdeckt, wartet der Roboter auf einen Auftrag (z. B. drei Tüten aus der mittleren Kiste). Die Kamera ermittelt mögliche Pickposen für die Tüten, der Roboter fährt (über mehrere Stützstellen, hier nicht dargestellt) die Pickpose an und der Vakuumgreifer wird aktiviert. Schlägt das Greifen fehl, so wird der Vorgang wiederholt. Ist das Greifen erfolgreich, fährt der Roboter zu einer Ablagepose und die Tüte wird auf dem Förderband platziert. Geht während der Fahrt die Tüte verloren (z. B. durch zu hohe Beschleunigungskräfte), so wird der Greifvorgang ebenfalls wiederholt. Nach erfolgreicher Ablage auf dem Förderband wird ggf. eine weitere Tüte gegriffen, solange bis der gesamte Auftrag abgearbeitet ist.

Anhand dieses Szenarios sollen nun folgende KI-gestützte Optimierungen durchgeführt werden:

- Steigerung der Pickrate (Wahrscheinlichkeit, dass erfolgreich gegriffen wird) durch Optimierung der Pickpose (Abschnitt 4.1.),
- Steigerung der Pickgeschwindigkeit (Verkürzung der Taktrate) durch vollautomatische Code-Optimierung mittels generativer KI (Abschnitt 4.2),
- Steigerung der Pickgeschwindigkeit durch interaktive Code-Anpassung mittels generativer KI (Abschnitt 4.3) und
- vollständig automatisierte Programmierung (Abschnitt 4.4).

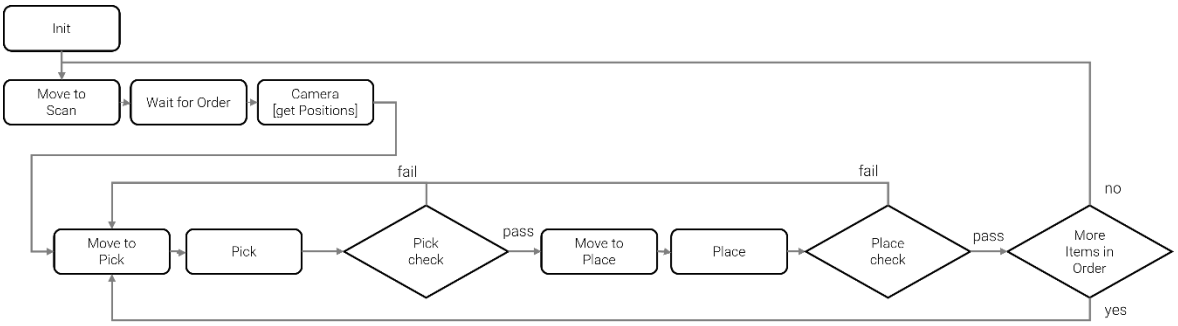

Bild 5: Vereinfachtes Ablaufdiagramm der Pick-&-Place-Zelle

4.1 Optimierung der Pickpose

Während des Betriebs der Zelle wird protokolliert, ob ein Pick erfolgreich war oder nicht. Die erfolgreichen bzw. nicht erfolgreichen Picks je Box stellen sich wie in Bild 6 gezeigt dar (grün: erfolgreich, rot: nicht erfolgreich). Die Visualisierung erfolgt schnell und einfach z. B. mittels des open source Tools Grafana.

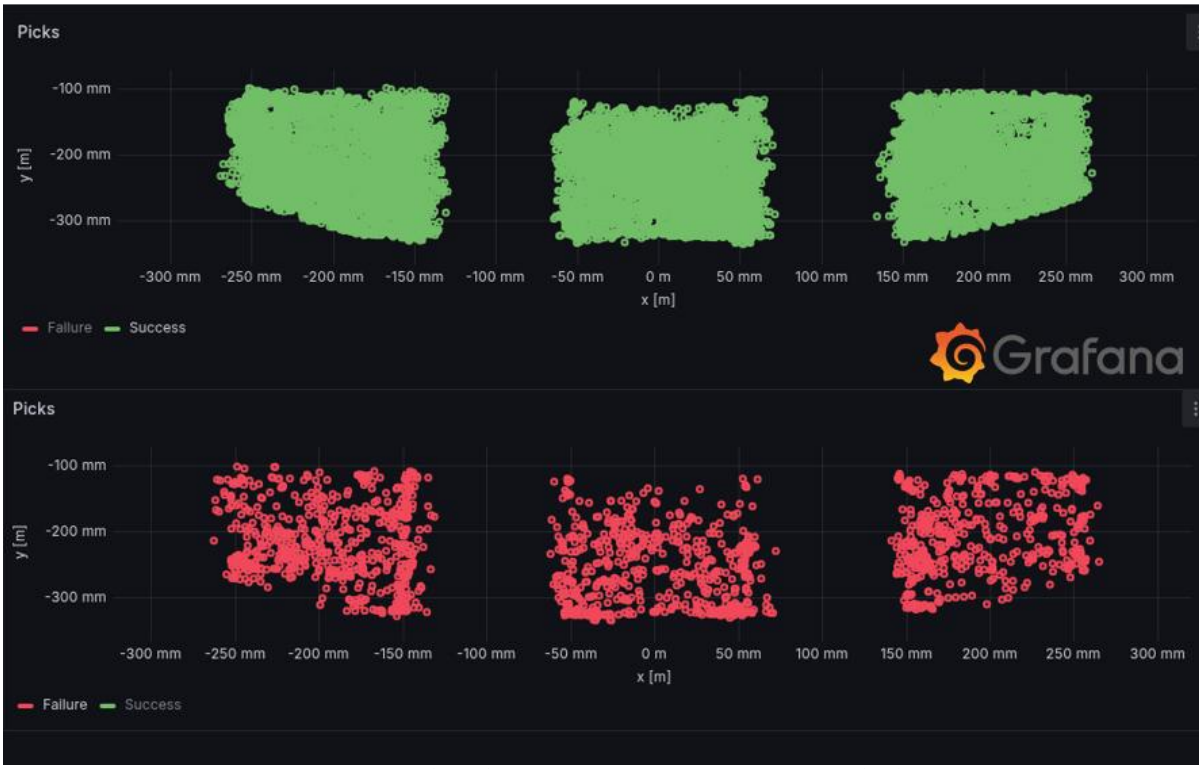

Bild 6: Erfolgreiche (grün) sowie nicht-erfolgreiche Picks (rot) je Box

Diese Information wird nun verwendet, um mit PyTorch (einer Python open source Programmbibliothek für das maschinelle Lernen) ein Modell zu trainieren, das zur Laufzeit für jede von der Kamera vorgeschlagene Pickpose (je Kiste werden bis zu zehn Posen detektiert) die Wahrscheinlichkeit für einen erfolgreichen Pick berechnet. Die Tüte mit der höchsten Wahrscheinlichkeit wird anschließend vom Roboter angefahren und gegriffen. So ist es möglich, die Pickrate (erfolgreiches Greifen von Tüten) von initial 95 % auf über 99 % zu steigern. Es kamen keine zusätzlichen Sensoren zum Einsatz –

lediglich die im Betrieb ohnehin anfallenden Informationen wurden systematisch ausgewertet und für das Training einer KI zur Regression eingesetzt.

4.2 Steigerung der Pickgeschwindigkeit durch vollautomatische Code-Optimierung

In diesem (und im folgenden) Beispiel soll von jeder Box je eine Tüte gegriffen und auf das Förderband gelegt werden. Ziel ist die Steigerung der Pickgeschwindigkeit, was sich ebenso wie die Erhöhung der Pickrate direkt auf die Wirtschaftlichkeit der Anlage auswirkt. Grundlage für die Optimierung ist eine Simulation der Zelle inklusive Applikation, die die wesentlichen Metriken zur Bewertung liefert (Testplan: Dauer einzelner Bewegungssegmente, Kollisionen, …), siehe Bild 7. Die Programmierung der Applikation erfolgt in Hochsprache (Python).

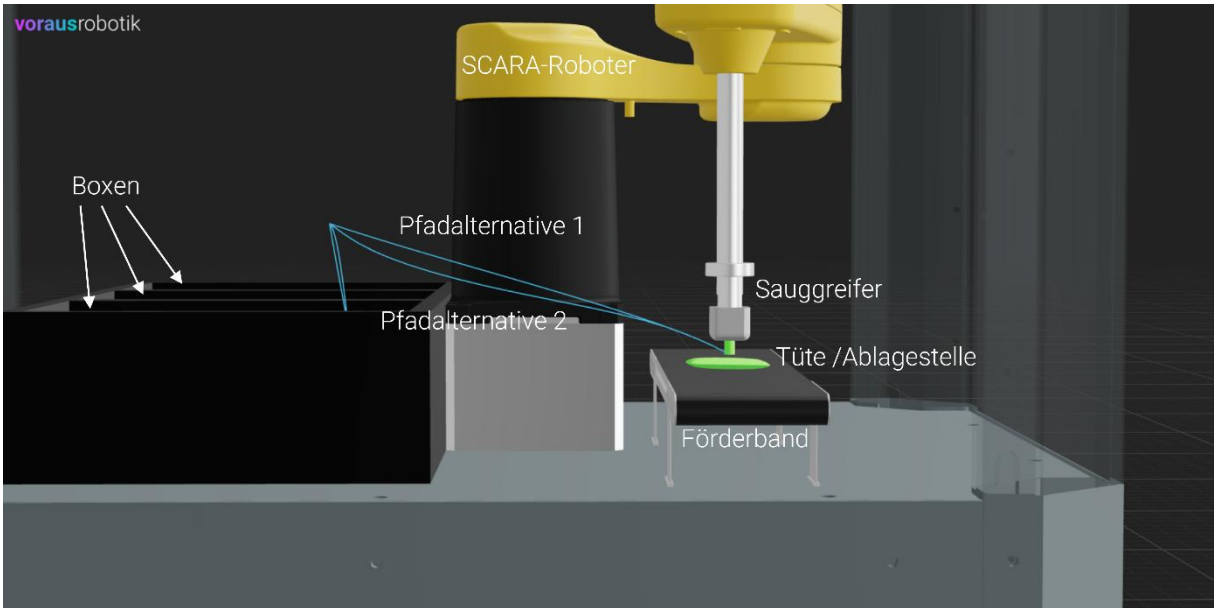

Bild 7: Simulierte Pick-&-Place-Zelle

Der GitHub CoPilot KI-Agent verwendete Gemini 3 Pro Thinking als LLM zur Code-Generierung. Durch die VSC-Integration (VSC: Visual Studio Code) hatte er Zugriff auf den ursprünglichen Python-Code der Applikation, die Software-Dokumentation von voraus robotik und mittels MCP (model context protocol) auf die von der Simulation gespeiste InfluxDB Datenbank (mit Informationen über Kollisionen, Pickdauer, erfolgreiche Ausführung des Auftrags etc.). Dadurch ist eine eindeutige Zuordnung von Simulationsergebnissen und zugehörigen Quellcode für jede Iteration der Optimierung gewährleistet.

Das Prompting beinhaltete unter anderem das eigentliche Ziel (Minimierung der Zeitdauer für den Pick-&-Place-Vorgang), welche Variablen angepasst werden dürfen sowie die Erstellung eines Testplans inklusive Erfolgskriterien.

Im sogenannten „Beast Mode“ ist der KI-Agent in der Lage, den generierten Code selbständig lokal in einem DevContainer auszuführen und anhand der Simulation dem

generierten Testplan zu unterziehen. Jede Iteration der Optimierung dauerte in etwa eine Minute auf einem handelsüblichen Rechner und beinhaltete unter anderem Starten der SW-Container, Simulationslauf und Reasoning. Die Gesamtdauer der Codeoptimierung betrug etwa 20 Minuten.

Am Ende wurde die KI Claude (Anthropic) gefragt, welche Änderungen der KI-Agent am Code vorgenommen hat. Das Ergebnis ist in Bild 8 gezeigt – so wurde unter anderem das Blending zwischen einzelnen Bahnsegmenten (Punkt 4) und die Synchronisation mit dem Förderband (Punkt 6) angepasst.

```
### Solution Implemented
Key changes in program_utils.py:
1. Increased HOVER_HEIGHT from 0.01m to 0.025m
2. Removed blending at critical pick/place hover positions (added .result() to wait
3. Increased linear movement speeds from 0.5x to 1.0x
4. Optimized blending percentages strategically:
   - 80% for initial approach to hover (safe open space)
   - 70% for OUT_OF_VIEW transitions (high-level safe movement)
   - 50% for post-pick return to hover (moderate safety)
5. Removed unnecessary sleep after pick operation
6. Optimized conveyor timing to 0.2s
```

Bild 8: Die vom KI-Agenten selbständig vorgenommenen Änderungen am Ablauf der Pick-&-Place-Applikation nach Claude

Im Ergebnis erzielte der KI-Agent eine Reduktion der Zeitdauer von 13,85 s auf 12,4 s, mithin also um 10 %.

4.3 Steigerung der Pickgeschwindigkeit durch interaktive Code-Optimierung

Im Gegensatz zum vorherigen Beispiel durfte der KI-Agent in diesem Fall den generierten Source-Code nicht selbst ausführen. Dieses Vorgehen entspricht dem üblichen Co-Pilot-Ansatz bei der Programmierung und erlaubt eine bessere Kontrolle des von der KI generierten Codes. Darüber hinaus ist es so möglich, eigene Vorschläge (und damit Expertenwissen) in den Optimierungsprozess durch Anpassung des Prompts einzubringen. Am Ende der ca. 40-minütigen interaktiven und iterativen Code-Optimierung stand eine Verkürzung der Taktzeit auf 11,6 s (anstelle von initial 13,85 s). Dies entspricht einer Verbesserung von etwas über 16 % und übertrifft im Ergebnis das vollständig autonome Vorgehen aus Abschnitt 4.2. Die Ursache dürfte in dem Einbringen von Expertenwissen in den Optimierungsprozess liegen, das letztendlich zu mehr Freiheitsgraden im Optimierungsprozess führt.

In beiden Fällen (Abschnitt 4.2 und Abschnitt 4.3) konnte und wurde der optimierte und in der Simulation automatisiert getestete Code ohne Änderungen auf die Anlage gespielt.

4.4 Vollständig automatisierte Programmierung

Die vollständig automatische Code-Erzeugung mittels generativer KI stellt aktuell die „Königsdisziplin“ dar. Betrachtet man die am Markt verfügbaren Lösungen, so stellt man fest, dass diese entweder auf Applikationsseite (noch) stark eingeschränkt sind oder eher eine natürlichsprachliche Schnittstelle zur Parametrierung existierender Abläufe implementieren. Auch wenn dies wichtige Schritte sind, so bleiben existierende Lösungen doch häufig hinter dem selbstformulierten Anspruch der Allgemeingültigkeit zurück. Die aktuellen Gründe sind in Abschnitt 2.2 dargelegt – insbesondere die mangelnde Berücksichtigung von Fehlerfällen und Edge-Cases sei hier nochmal erwähnt.

Auch wir waren mit der automatischen Code-Generierung für die gesamte Pick-&-Place-Zelle mit vertretbarem zeitlichen Aufwand und der für den industriellen Einsatz erforderlichen Qualität nicht erfolgreich – unabhängig vom verwendeten LLM (ChatGPT, Claude, Gemini und Grok). Was jedoch sehr gut funktionierte, war die Generalisierung eines Minimalbeispiels: Es wurde der Pick-Vorgang für eine Kiste programmiert und anschließend mittels KI auf die weiteren Boxen generalisiert. Bild 9 zeigt den manuell geschriebenen Programmcode (links) und den von der KI erzeugten Code für die mittlere Box (rechts). Die Unterschiede sind farblich hervorgehoben und verdeutlichen, dass das LLM in der Lage war, die Indizierung der Kisten zu verstehen und anzupassen.

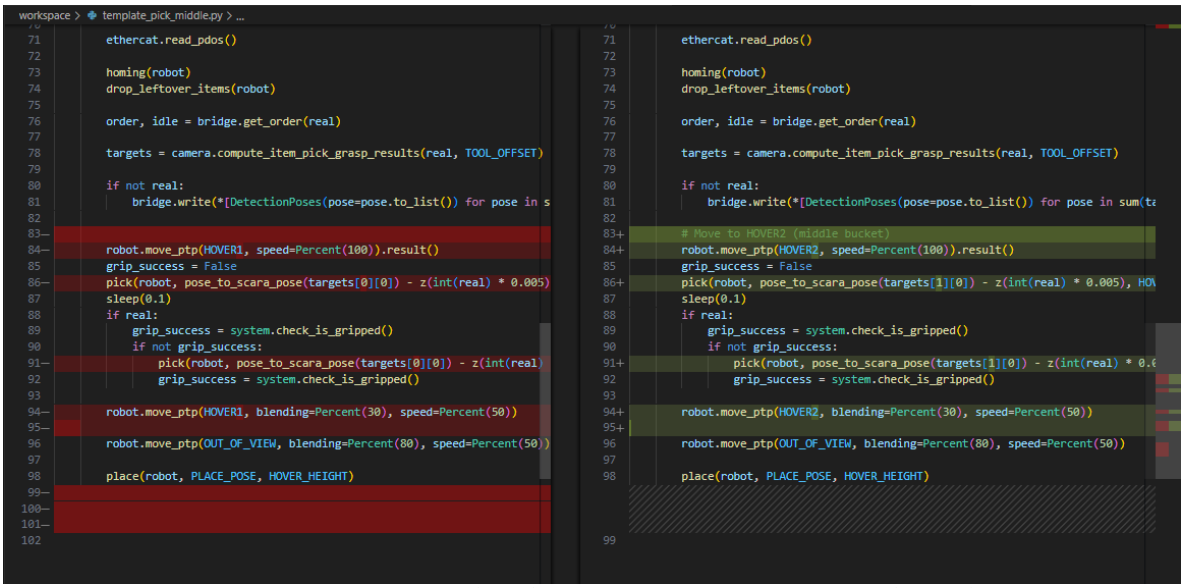

Bild 9: Minimalbeispiel (links) und Erweiterung durch die KI (rechts)

Nach simulativer Absicherung im Rahmen der Industrial DevOps-Pipeline wurde dieser Code ebenfalls direkt und automatisiert auf die reale Anlage aufgespielt und ausgeführt.

5. Fazit und Ausblick

Am Beispiel einer industriellen Pick-&-Place-Zelle (mit marktüblichen, heterogenen Komponenten) wurde dargelegt, welchen praktischen Nutzen und wirtschaftlichen Mehrwert KI-Agenten jetzt schon entfalten können. Dabei zeigt sich die Mächtigkeit von KI-Agenten zur Optimierung, wenn wir geschickt (generative) KI, Simulation und reale Zelle kombinieren. Fassen wir die Ergebnisse kurz zusammen: Die Pickrate konnte von 95 % auf über 99 % gesteigert werden, wobei hier die Daten der realen Zelle verwendet wurden. Durch den Einsatz generativer KI wurde die Pickgeschwindigkeit um mehr als 16 % erhöht. Grundlage war eine Simulation der Zelle und ein (vom KI-Agenten) generierter Testplan.

Betrachten wir die zeitlichen Aufwände für die Code-Optimierung in Abschnitt 4.2 und Abschnitt 4.3 so liegt dieser selbst bei konservativer Schätzung deutlich unter einer Stunde. Die Aufwände zum Aufsetzen des KI-Agenten beschränken sich im Wesentlichen auf Simulation und Testpipeline, die wiederum integraler Bestandteil des Industrial DevOps Frameworks sind. Setzen wir für den Mehraufwand zusätzliche drei Stunden an, so nahm die Optimierung insgesamt einen halben Tag in Anspruch. Umfangreiche Tests vor und während des Deployments entfallen vollständig. Insofern ist eine Reduktion der zeitlichen Aufwände im Vergleich zum heutigen Vorgehen (manuelle Optimierung, Aktualisierung verschiedenster Steuerungen der realen Anlage, verbunden mit umfangreichen Tests) um bis zu 80 % realistisch.

Das Beispiel zur Codegenerierung (Abschnitt 4.4) unterstreicht deutlich, dass der Aufwand für das Prompting umso geringer ausfällt, je klarer umrissen die Aufgabe ist. Hier liegt der wesentliche Schlüssel zur Zeitersparnis: Ziel muss nicht zwingend die automatische Programmierung der gesamten Applikation sein – ausgehend von einem manuell erstellten Minimalbeispiel scheint es vielmehr effizienter dieses zu generalisieren.

Sowohl Abschnitt 4.3 als auch Abschnitt 4.4 zeigen, dass das Einbringen von Expertenwissen die Leistung eines rein KI-basierten Ansatzes deutlich zu steigern vermag. Dies scheint der momentane „sweet spot" der KI-gestützten Entwicklung zu sein – Experten stellen ihr spezifisches Wissen zur Verfügung und die KI entlastet von zeitraubenden Routinetätigen.

Was waren weitere Erkenntnisse? Das Aufsetzen der Testinfrastruktur (Simulation und automatisierte Auswertung) ist für den erfolgreichen Einsatz von KI-Agenten entscheidend. Eine durchgängige Toolchain, wie sie bei konsequenter Anwendung des Industrial DevOps-Ansatzes ohnehin entsteht, ist hierfür eine hervorragende Grundlage und reduziert den Aufwand deutlich. Die Programmierung in gängigen Hochsprachen (Python, Rust, …) stellt sicher, dass existierende KI-Frameworks problemlos und einfach eingebunden werden können. In allen vorgenannten Fällen sind die (automatisiert getesteten) Ergebnisse mit geringem Risiko und in kürzester Zeit auf die echte Zelle übertragbar – dank einheitlicher Code-Basis.

So liefert KI bereits heute einen echten Mehrwert für die industrielle Automation – jenseits des aktuellen Hypes.

6. Unternehmensinformationen und Kontaktdaten

Weitere Informationen zu unserer Softwareplattform und wie wir Sie beim Einsatz von KI in der industriellen Automatisierung unterstützen können finden Sie auf unserer Homepage www.vorausrobotik.com. Unsere Dokumentation mit vielen Beispielen gibt es hier: https://docs.vorausrobotik.com/.

Teile der Fallstudie werden vom Bundesministerium für Wirtschaft und Energie (BMWE) aufgrund eines Beschlusses des Deutschen Bundestages gefördert (Projekt ALIGMO, KK5640702GR4).

Unsere Kontaktdaten

voraus robotik GmbH | Carl-Buderus-Str. 7 | 30455 Hannover | **voraus**robotik.com

Ansprechperson

Tobias Ortmaier | tobias.ortmaier@vorausrobotik.com