THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Towards an infrastructure for preparation and control of intelligent automation systems

Perspectives and lessons learned

Endre Erős

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Towards an infrastructure for preparation and control of intelligent automation systems

Perspectives and lessons learned

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Printed by Chalmers Reproservice
Gothenburg, Sweden, January 2022

To my lovely and supporting family, friends, and colleagues.

Towards an infrastructure for preparation and control of intelligent automation systems

Perspectives and lessons learned

Endre Erős
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In an attempt to handle some of the challenges of modern production, intelligent automation systems offer solutions that are flexible, adaptive, and collaborative. Contrary to traditional solutions, intelligent automation systems emerged just recently and thus lack the supporting tools and infrastructure that traditional systems nowadays take for granted. To support efficient development, commissioning, and control of such systems, this thesis summarizes various lessons learned during years of implementation.

Based on what was learned, this thesis investigates key features of an infrastructure for modern and flexible intelligent automation systems, as well as number of important design solutions. For example, an important question is raised whether to decentralize the global state or to give complete access to the main controller.

Moreover, in order to develop such systems, a framework for virtual preparation and commissioning is presented, with the main goal to offer support for engineers. As traditional virtual commissioning solutions are not intended for preparing highly flexible, collaborative, and dynamic systems, this framework aims to provide some of the groundwork and point to a direction for fast and integrated preparation and virtual commissioning of such systems.

Finally, this thesis summarizes some of the investigations made on planning as satisfiability, in order to evaluate how different methods improve planning performance. Throughout the thesis, an industrial material kitting use-case exemplifies presented perspectives, lessons learned, and frameworks.

Keywords: Intelligent automation, architectures, virtual preparation, virtual commissioning, planning as satisfiability.

List of Publications

This thesis is based on the following publications:

[A] **Endre Erős**, Martin Dahl, Atieh Hanna, Per-Lage Götvall, Petter Falkman and Kristofer Bengtsson, "Development of an Industry 4.0 demonstrator using Sequence Planner and ROS2". Robot operating system (ROS): The complete reference. vol. 5, pp. 3–29, Springer, 2021.

[B] **Endre Erős**, Martin Dahl, Atieh Hanna, Anton Albo, Petter Falkman and Kristofer Bengtsson, "Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system". IEEE International Conference on Emerging Technologies and Factory Automation vol. 24, pp. 407–413, ETFA 2019.

[C] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Evaluation of high level methods for efficient planning as satisfiability". IEEE International Conference on Emerging Technologies and Factory Automation vol. 26, ETFA 2021.

[D] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Towards compositional automated planning". IEEE International Conference on Emerging Technologies and Factory Automation vol. 25, pp. 416–423, ETFA 2020.

Other publications by the author, not included in this thesis, are:

[E] **Endre Erős**, Martin Dahl, Kristofer Bengtsson, Atieh Hanna and Petter Falkman, "A ROS2 based communication architecture for control in collaborative and intelligent automation systems". *Procedia Manufacturing Volume 38, 2019, Pages 349-357*, 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM), Limerick, Ireland 2019.

[F] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Virtual preparation and commissioning of intelligent automation systems". *To be submitted to: Robotics and Computer-Integrated Manufacturing, 2022.*

[G] **Endre Erős**, Martin Dahl, Petter Falkman and Kristofer Bengtsson, "Investigating architectures for intelligent automation systems". *To be submitted to: Robotics and Computer-Integrated Manufacturing, 2022.*

[H] Martin Dahl, **Endre Erős**, Atieh Hanna, Kristofer Bengtsson, Martin Fabian and Petter Falkman, "Control components for Collaborative and Intelligent Automation Systems". *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Volume 24, Pages 378-384, 2019.*

[I] Martin Dahl, **Endre Erős**, Atieh Hanna, Kristofer Bengtsson, Petter Falkman, "Sequence Planner - Automated Planning and Control for ROS2-based Collaborative and Intelligent Automation Systems". *arXiv 1903.05850, cs.RO, 2019.*

[J] Atieh Hanna, Kristofer Bengtsson, Martin Dahl, **Endre Erős**, Per-Lage Götvall and Mikael Ekström, "Industrial Challenges when Planning and Preparing Collaborative and Intelligent Automation Systems for Final Assembly Stations". *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Volume 24, Pages 400-406, 2019.*

[K] Elinor Jernheden, Carl Lindström, Rickard Persson, Max Wedenmark, **Endre Erős**, Sabino Francesco Roselli and Knut Åkesson, "Comparison of Exact and Approximate methods for the Vehicle Routing Problem with Time Windows". *IEEE International Conference on Automation Science and Engineering (CASE), Volume 16, Pages 378-383, 2020.*

Acknowledgments

This research is supported by Chalmers University of Technology, Volvo GTO, and VINNOVA under the projects Unification (contract nr. 2017-02245) and Unicorn (contract nr. 2017-03055).

Acronyms

AI:	Artificial Intelligence
BDD:	Binary Decision Diagram
SP:	Sequence Planner
ROS:	Robot Operating System
VC:	Virtual Commissioning
IVPC:	Integrated Virtual Preparation and Commissioning
PLC:	Programmable Logic Controller
SAT:	Propositional Satisfiability
SMT:	Satisfiability Modulo Theories

Contents

Part I

Overview

CHAPTER 1

Introduction

1.1 Automation is changing

The true advance of automation is not marked by putting new products on the market, but by producing faster with greater variability and quality. Customizing a product online and expecting it to arrive within days is becoming the norm. Of course, manufacturers are starting to grasp the potential of this customer oriented market, however they face some new challenges [1]. Not only do companies have to re-think the way they manufacture, but in order to turn a profit and beat the competition, they have to ensure that the costs of unique product manufacturing remain similar to the costs of mass production [2].

From a high-level perspective, this represents a need to control the value chain in real-time [3]. In order to do so, different discrete activities of the value chain have to be more connected, relevant information from all segments of the value chain has to be transparent, and most importantly, customers and business partners have to be integrated into the production processes [4].

From a low-level perspective, this demands flexible and adaptable production systems which can not only change the pace of production, but also the

type of product, in real-time [5]. Developing automation solutions for such production systems is fundamentally different from existing traditional methods, and as such, existing tools and frameworks can not be directly utilized. Hence, this thesis aims to lay down at least some of the groundwork needed to develop new, *intelligent* automation solutions of the future.

1.2 Challenges

An industrial use-case from Volvo Trucks is used as the main illustrative example throughout the thesis. Being one of the biggest truck manufacturers in the world, Volvo aims to produce enough trucks with compelling properties in order to maximize its share of the market. As such, they face all challenges of modern production.

Compared to today's standards, trucks that were manufactured until 1990's did not have a substantial variability and complexity. They relied mostly on mechanical components which were standardized across several models. This meant that the manufacturing complexity increased at a reasonable rate and could be managed by traditional production and automation solutions. However, the manufacturing complexity is now increasing at a much higher rate.

To begin with, the product itself is one of the factors that contributes to the increase in manufacturing complexity. The introduction of new electronic, safety, sensor and software systems has transformed the relatively simpler truck of the 90's into the necessarily complex truck of today.

Another factor is the increase in product variability. Customers often have unique requests, and in order to satisfy as many as possible, products have to be unique as well. This often calls for modified components which can not be standardized across many product variants anymore. As such, individual product components are also seeing an increase in variability.

Yet another factor is the introduction of new product types, such as hybrid, battery powered, hydrogen fuel-cell, and autonomous trucks. Traditionally, new production facilities would be built for new product types, but the current demand volume of each individual type makes this option economically infeasible. A mixed-model assembly approach has to be taken instead, which means that various types of products are manufactured on the same assembly line.

Beside these challenges, manufacturers face a myriad of other issues which are more or less tied to the product itself. For example, in order to alleviate the need to store large quantities of products which have to be available for the customer any time, manufacturers aim towards more sustainable production where products are manufactured only in the required quantities and only when needed. The challenge here is the tight connection with the customers.

Yet, the bulk of manufacturing today is not ready to make a complete transition towards intelligent and sustainable production. As many things in life, the change in demand is not happening overnight. Companies do not only face the challenge of how to keep up with the competition and deliver more specialized products to the market, but also how to satisfy the current market's need for products it can produce on a mass scale using current traditional methods.

Before we try to find answers to some of the questions that came up, let's summarize the main challenges here:

1. *Variability*: In order to satisfy a larger share of the market, companies have to offer products which are specialized and customizable.

2. *Speed*: The market is moving and changing fast, and in order to keep their customers happy, companies have to develop, produce and deliver their products in time.

3. *Cost*: Variability and speed wouldn't be a issue if there wasn't a constraining factor such as cost. Not only do companies have to put custom products on the market fast, but also at a price which can be met by a big share of the market.

4. *Society*: Companies have to take care of the society they impact and influence. This does not only involve workers, but also customers.

5. *Sustainability*: There shouldn't be a need to manufacture and store large quantities of products if production can be made more sustainable. This means production on demand, and is considers producing only the items which are demanded, at the right time and quantity.

6. *Legacy*: Until the market has changed in such a way where it is economically feasible to build new and streamlined production facilities, current production has to keep on going side by side with new production methods.

1.3 A traditional approach

A question that is often asked is why not use traditional production solutions, where industrial robots, feeders, and conveyors pick and deliver the necessary material to the autonomous platforms. After all, such solutions have been working until now, so why should it be different this time?

There are two main reasons why traditional production solutions are not suitable anymore: spatial and economic. The first reason is due to the necessary space to implement such a solution and the safety issues that go along with it. Traditional solutions which comprise of industrial robots, feeders, and conveyors are very space demanding. The main reason for this is the need for such equipment to be caged or protected, in order to assure that serious injuries and death are avoided all the time.

The economic reason encompasses several other challenges such as flexibility, implementation, and maintenance. Namely, it is economically infeasible to implement and maintain a traditional production solution which is flexible enough to handle high variability and the constant need for updating and maintaining the systems due to new changes. Flexible systems have to be virtually free to reconfigure and update, both in terms of time and money.

In order to have truly flexible systems, control software has to be intelligent enough to handle all sorts of changes in the system in real-time. However, it is hard to advocate for such a drastic change since programming explicit control code has a strong tradition in industry. As proven over the decades, it looks like it is a sufficient and easy way of programming behavior for robust, fully automatic, and topologically static automation solutions.

However, it is becoming very hard, or rather impossible, to implement the required behavior of production systems which can fulfill modern requirements using traditional automation solutions. Such production systems have to be flexible, collaborative, and easily reprogrammable, and current automation tools are just not applicable. It is time to look at another approach, which might be the answer to some of these challenges.

1.4 An intelligent approach

As it is already hinted at, we need solutions which can be easily moved around, reconfigured and reprogrammed. More importantly, we need solutions which are enabling operators instead of restricting them. So, what can we do?

We can start with looking at collaborative robots. Contrary to their industrial counterparts, collaborative robots are equipped with sophisticated sensors and software which enable them to react to external forces in real time. This feature has several benefits. It enables operators to safely work in close proximity to collaborative robots, thus, caging and allocating a lot of space for safety reasons is not needed anymore. Moreover, this feature enables the operators to easily re-program and re-position the robots, keeping them up and running without the need to stop production and call in specialized robot programmers.

The next thing we should look at are vision systems. In order to move robots around, reconfigure workstations, localize different parts, detect tags, and keep track of operators, we have to use different kinds of cameras, structured light scanners, light detection and ranging sensors, etc. For example, a set of ordinary IP cameras above a workstation can report a rough estimate of large items in the station such as crates, tables, or AGVs, using some sort of a tag detection algorithm. Mounted at critical positions, or, for example, on the face plates of robotic manipulators, structured light scanners can detect and localize specific items in the workstation on a sub-millimeter precision level. At last, dedicated camera and lidar systems can assure operator safety in workstations by, for instance, reducing the velocities of nearby AGVs or robotic manipulators.

Hardware is just one part of the solution, so the next thing we have to investigate are intelligent control frameworks, communication architectures, and supporting software tools. As it will be discussed in this thesis, our view is that programming explicit control code just won't cut it for complex, flexible and collaborative systems. The idea is to apply a model-based approach instead [6], with accompanying tools such as automated planning, formal verification, virtual preparation and commissioning, as well as testing. Additionally, we have to ensure a robust and reconfigurable way for different decentralized pieces of hardware and software to work together and communicate.

Another important thing that we have to look at is the compatibility with legacy systems and other traditional automation solutions. Replacing old or

incorporating new systems into the existing manufacturing process can be quite challenging, as it is often the case that interfaces have to conform to the standards of legacy systems and not the other way around. Other than that, equipment with dedicated industrial controllers has to be accessed using different kinds of communication interfaces.

Finally, intelligent production solutions need operators, and again, there are several reasons for this. Tasks that are performed in a workstation can differ a lot, for instance, some are very easy and repetitive and some are executed once every so often and are complex or very tactile. Designing robotic solutions which can always handle 100% of problems is futile, so tasks should be distributed between robots and operators in such a way that an optimal flow is achieved. This could mean that the robot executes 90% of repetitive and injury prone tasks, while the operator executes a few tactile operations which would be very hard for the robot to successfully perform. Moreover, operators would aid the robots when necessary, for example with picking that very last tricky item that got stuck in the corner of the box.

As briefly mentioned earlier, operators would have additional tasks such as re-configuring and maintaining workstations. Because of the high flexibility of intelligent solutions, workplaces now have the ability to change their topology in order to optimize for a specific manufacturing scenario. For example, when a different type of truck is to be assembled, the material facade for that type can be moved manually to the robot so that it can pick the needed material for that truck type. The ease of re-positioning robots and re-teaching poses, allows the operators to handle and maintain the workstations in real time without the need to halt production.

1.5 Research approach

The research during my studies is not driven by hypotheses, but a collective vision and a multitude of industrial applications. You could look at this vision as a set of research questions, even if they are really only being formulated for the purpose of having research questions in this thesis. One could say that this main vision makes this research approach inductive, even if it tends to move around and change shape more often than we actually try to write about it. That being said, I am not sure how many of such objectives have we actually achieved and how many just faded into the abyss.

1.6 Research questions

In order to shape and guide this thesis, let's formulate several research questions. These questions will be addressed throughout the thesis, however, the reader can expect a short and explicit summary at the end of the thesis.

RQ1 Why do we need intelligent automation systems?

Over the decades, traditional automation has proven itself as a robust way to fully automate simple, static, predictable, and repetitive tasks. However, tasks defined by new production challenges have to be collaborative and dynamic, which also often makes them unpredictable and complex. Applying traditional solutions in order to automate such tasks is becoming very hard, so we need to look for other methods. This is where intelligent automation might be an answer to a part of the problem.

RQ2 What to consider when implementing architectures for intelligent automation systems?

The ease of implementation, performance, and even behavior of a system is tightly coupled to the architecture it implements. Instead of ending up with an ad-hoc architecture that might not fulfill the specified system requirements, resource, control, and communication architectures, as well as their integration, have to be investigated.

RQ3 How to support the development of intelligent automation systems with tools such as virtual preparation, commissioning, and testing?

Intelligent automation might be the answer to some of the mentioned production challenges. However, being a novel approach, it lacks supporting tools and methods which mature and reliable technologies nowadays take for granted. Software for design, simulation, commissioning, control and maintenance of intelligent automation systems either doesn't exist or it is in infant stage of development. For example, industry leading software tools for traditional automation systems like Process Simulate and Delmia include support for simulation and virtual commissioning [7]. However, neither of those solutions is applicable to intelligent automation systems, nor does other software exist which can support development, simulation and virtual commissioning of such systems.

RQ4 *What makes intelligent automation systems intelligent and how can we make intelligent reasoning fast enough?*

Intelligence has different aspects, ranging from learning, memory and knowledge, all the way to planning, creativity and decision making. For example, [8] defines intelligence as a very general mental capability that involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly, as well as learn from experience. As described in [6], intelligent automation systems are *adaptive*, *flexible*, *deliberative*, *reactive* and *learning*. However, in order to keep the title *intelligent*, they have to adapt, react, and deliberate in due time. If possible, they have to be able to do that in real-time.

1.7 Research journey

I was quite lucky to begin my research journey somewhere in March 2017, when Per-Lage introduced me to one of his robotics projects within Volvo and told me to implement a remote ROS master. I remember it like it was yesterday because I just nodded while having no idea what he was talking about. Thankfully, he understood that and gave me a book [9], which was my first introduction to ROS and the world of robotics.

My tasks were to develop and implement different scenarios that would demonstrate the possible utilization of collaborative robots and autonomous transport robots in the industry. We called our joint effort the Collaborative Robot Station Laboratory, or shortly, the CRS-lab. While implementing those demos, I got to play around and learn about ROS, communication, computer vision, linux and robotics. Eventually, when it was time to make everything work together, the part where I got stuck was *control*. I spent so many hours implementing behavior for these systems, and they never worked exactly as I would like them to. There had to be something that I was missing.

It turned out that I was trying to explicitly implement all the possible behavior using "if-then-else" programming. Looking back at it now, it was a miracle that anything even moved.

Luckily, I got introduced to Patrik's thesis [10], where he had a very nice example about extended finite automata called the *advanced stick picking game*. It is a game for two players, where each of them pick sticks in alternating order with some extra rules. The loser is the player to pick the last stick.

Before I have read the thesis, I have known about state machines, but there was something new for me in this example. I was introduced to uncontrollable events and automata synchronization.

When I finally got my hands on Supremica [11], I realized that I could model all the components of my system separately and then just synchronize them to get a big transition table which contained all the behavior that I failed to explicitly program. I wrote a simple runner that goes through these transitions and suddenly things were much easier to implement and something finally actually worked! As a tribute, I eventually implemented a version of the stick picking game for a human operator and a collaborative robot.

After a while, we realized that there is a lot of potential for using ROS in the industry, so I joined the Unification project [12]. The goal of this project was to investigate the possibility to transform an existing manual engine assembly station into a collaborative workstation. This project is part of Volvo's global effort to develop factories of the future that can handle the challenges of modern production requirements [13]. Some of the main traits this station should have is resource flexibility and adaptability, which means that operators and humans should be able to execute the same tasks, even using the same tools. Moreover, it should be easy for the operators and robots to either work alone, or side by side. During this project, we learned that in order to achieve the desired flexibility, vision, safety, and keeping track of where things are in the world is crucial.

In October 2018, I got quite lucky to be able to continue my research journey at Chalmers, at the Automation research group. As an extra layer of luck, I got assigned the same project that I had when leaving Volvo, Unification. It meant that I could continue building on top the things I have already worked with, with people I have already worked with, Kristofer and Martin. Together, we worked on Sequence Planner and its accompanying tools, and by the time Unification ended in October of 2020, we have managed to publish several papers about communication, control, and virtual commissioning.

During the same period, I was part of another project together with Martin and Petter, called Unicorn [14]. This project envisioned automatic refuse collection and sorting systems, utilizing a fleet of small autonomous robots that continuously collect refuse from smart household refuse bins. This refuse would then be transported to a large centralized sorting station which in turn is emptied on demand by a truck of the correct type. A part of our job was

to investigate and develop a coordination and scheduling control system for this robot fleet.

Unfortunately, when the time came to implement a demonstration using the actual robots in the neighbourhoods, Covid-19 put a hard halt on everything. Nevertheless, we continued to build a simulation environment where we would test if Sequence Planner could be used to control a fleet for up to 6 robots. It turned out that it could. This project helped us publish a few papers about planning and control. We have also learned that coordinating a fleet of robots is far from trivial, even with help from state of the art tools like [15].

In 2021, the last project I was part of was Robot-in-the-air, or shortly Rita. It is the use-case that I am using as an example in this thesis, so I won't write too much here. In essence, it is one of the first steps towards a completely reconfigured material handling approach, which is also a part of Volvo's vision of factories of the future.

1.8 Contributions

This compilation thesis summarizes and builds upon work from several papers. The main topics of this thesis are architectures, planning performance, and preparation and commissioning of intelligent automation systems. Every topic is addressed by paper or two, which are appended at the end of the thesis.

Paper A starts the discussion about architectures and the fundamental differences between centralized and distributed state. It also contains some information about supporting tools for preparation and virtual commissioning which we learned while working with the Unification project. Chapter 3 dives deeper into the problem of architectures in intelligent automation systems and provides a spectrum of implementation possibilities when developing architectures of such systems. As such, these contributions aim to address *RQ2*.

Paper B investigates the possibility to utilize virtual commissioning when developing intelligent automation systems, in order to ease engineering work and shorten the actual commissioning time. Chapter 4 builds upon that idea and describes in detail an actual procedure of how to perform virtual commissioning of a real intelligent automation system. Virtual preparation and commissioning is a big part of developing intelligent automation systems and working on these tasks will help answer *RQ3*.

Paper C investigates which algorithms increase planning performance, as

it evaluates and compares several high level planning methods on a set of standard benchmarks. It focuses on planning as satisfiability as the leading approach for solving difficult planning problems. Paper D addresses the performance challenge by presenting a compositional approach based on abstraction refinement that iteratively generates, solves and composes partial solutions from a parameterized planning problem. It shows that this approach decomposes the monolithic planning problem and thus significantly speeds up plan calculation, at least for a class of tested planning problems. These attempts address *RQ4*.

1.9 Outline

This thesis has two parts organized in the following way. Part I presents an overview of the work done, while Part II contains the appended papers. Part I has eight chapters. Chapter 1 introduces the challenges of modern production and proposes a new way of addressing them, namely intelligent automation systems. Chapter 2 presents an industrial use-case, as well as some tools and prerequisites that are going to be used throughout the thesis. Chapter 3 investigates resource, communication, and control architectures, while Chapter 4 presents a framework for virtual preparation and commissioning of intelligent automation systems. Chapter 5 summarizes investigations made on planning as satisfiability, as well as a compositional planning approach. Chapter 6 concludes the thesis and presents ideas for future work. In Chapter 7, additional discussions about modern automation are appended. Chapter 8 summarizes the appended papers.

CHAPTER 2

An industrial use-case

Material handling is a crucial process of logistics and manufacturing which occurs at the shop floors, warehousing, distribution, and disposal. It concerns the short-distance movement of material inside the manufacturing facilities such as transportation from storage to the assembly line, and it can involve other things such as material protection and storage [16].

At Volvo Trucks, material transportation to the assembly line is currently done in two stages. Firstly, the material is gathered from storage and brought to re-stock the material facades, which are accessed by operators in pre-assembly and kitting stations. When a pick order arrives, pick-to-light systems enable operators to quickly gather the ordered material and place it on a trolley which is later delivered to assembly stations nearby.

The increased material handling complexity introduces several problems, two of which are mentioned here. The first problem is that, since there is so much more material variability, there is also much more material that has to be stored, transported and picked. Introducing new variants of components to be picked along existing material will soon make the kitting stations run out of space. The second problem is that, since there is such a big increase in material handling complexity, operators will have a hard time keeping the

process fast, safe and error free.

Material handling can thus be recognized as one of the main bottlenecks for a mixed-model assembly approach. An option that Volvo Trucks is considering is to completely reconfigure the way that material handling is currently executed. Namely, the idea is to disconnect the material handling from the assembly area and move it to another place dedicated completely to such operations. At this place, operators and collaborative robots would carry out order picking, kitting, and pre-assembly. The prepared material would then be transported through the facility by autonomous platforms and delivered to the right assembly station, just in time.

2.1 Collaborative kitting

This work focuses on a kitting use-case, being such a crucial part of material handling. Kitting is the process of collecting required material into a single *kit* which is then transported to wherever it is needed. It is currently performed by operators, utilizing a pick-to-light system. When the order arrives to the station, the operators have several minutes to assemble a kit onto a trolley based on the order, after which the trolley is moved out of the kitting station.

As an initial step towards a redesigned material handling approach, our job was to investigate how a kitting station could be upgraded to support the operators with picking the ordered material. In order to be flexible, save space, and allow operators to move around safely, we chose to put a collaborative robot on a tall gantry and named the solution *Robot-in-the-air*, or shortly, *Rita*.

The nice thing about this solution is that the gantry provides a 7th degree of freedom to the robotic manipulator, increasing its workspace to cover the whole material facade. Another great detail is that the gantry is mounted on wheels, allowing the operators to move it around in the station and in-between stations if necessary. Moreover, the solution allows the material facade to be pushed below the gantry, which means that the solution does not occupy additional area in the station.

The material facade, as it can be seen on the left side if Figure 2.1, is holding a number of blue bins in which material is stored. For a specific assembly scenario, the material that has to be picked from those bins is defined in a *picking order*. In order to pick different types of items from the material

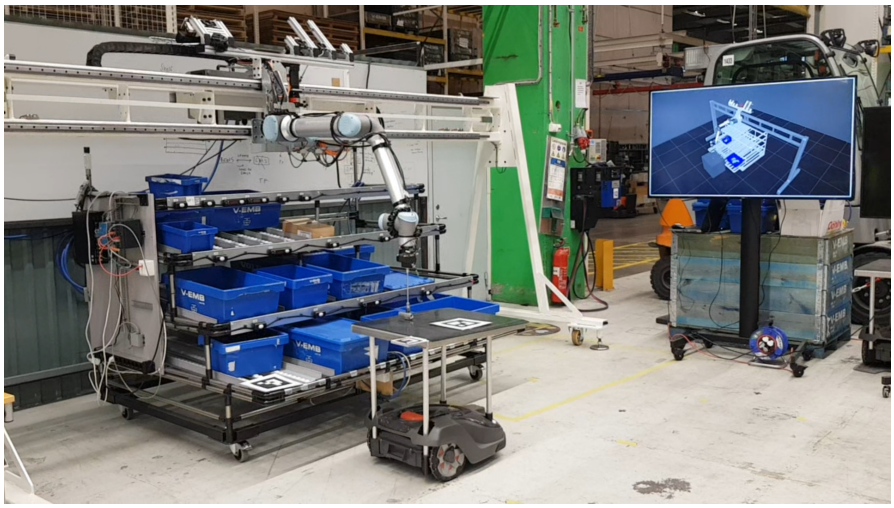


Figure 2.1: Robot-in-the-air material kitting solution.

facade, the robot can choose a picking tool that is best suited for that item. The tools are always accessible to the robot as they are mounted on a rack which travels together with the base of the robot on the gantry. The swift tool change is enabled by pneumatic *rsp* connectors.

Together with the picking tools, a battery powered structured light scanner is mounted on the tool rack. While mounted, it is constantly charging so that it can be ready for operation when mounted by the robot. The scanner is then used to scan and detect items that are to be picked from the blue bins.

The *Rita* solution is required to prepare a kit in time, based on the received order. If the robot is unable to pick every item in the order list, an operator should finish the kit utilizing the pick-to-light system. The kit is assembled on an autonomous platform which brings the kit to its designated assembly station at the right time. In order to implement this solution, we developed a preparation, commissioning and control infrastructure based on ROS and its accompanying tools.

2.2 A framework based on ROS

During the past decade, various platforms have emerged as middle-ware solutions trying to provide a robotics common ground for integration and communication. This effort was fuelled mainly by the reason that a lot of time was spent on re-implementing the software infrastructure required to build complex robotics algorithms. Organizations would spend the bulk of their time dedicated for a project in re-writing drivers for sensors and actuators, and re-inventing different infrastructures to enable communication between different resources and programs. In the end, too little time was left to implement the actual intelligent robotics programs based on this developed tooling. Even inside the same organization, the re-invention of drivers and communication systems was re-implemented for each new project.

This is where Robot Operating System (ROS) [9] emerged to put a stop to the need to constantly re-invent the wheel. With its large and enthusiastic community, ROS enables all users to leverage the power of a shared library of tools, algorithms and simulation packages. Moreover, ROS nowadays integrates the scalable, robust and well-proven Data Distribution Service (DDS) as its communications layer, enabling ROS to be used in industrial and mission critical scenarios.

Systems based on ROS rely on a set of *nodes* which communicate between themselves in order to achieve a desired collective behavior. Segmenting code into nodes which execute different specific tasks, enables users to maintain a good structure, as well as to distribute code on different machines. For example, if a specific piece of software has to run on a dedicated machine, wrapping it in a ROS node will enable this software to be integrated into the network and used in conjunction with other software and ROS based tools like *tf*[17], *moveit*[18] and *rviz*.

TF

In the Rita use-case, we heavily utilize *tf* in order to define and prepare scenarios, keep track of where things are in the world, and calculate relationships between those things. In a nutshell, it is a ROS package that lets the user keep track of multiple coordinate frames over time. It does so by maintaining relationships between coordinate frames in a tree structure buffered in time, and lets the user manipulate transforms between any two coordinate frames

at any desired point in time.

Usually, a global *tf* tree is keeping track of all things in a scenario, however, in some cases it could be beneficial to have multiple trees to keep track of frames in certain subsystems. In our use-case, a global *tf* tree maintains frames and relationships which can be directly measured, as well as those which can only be estimated.

2.3 Control

An intelligent automation system has resources. In order to achieve a desired state in the system, an overall control scheme has to choose the correct state for those resources. At the same time, the control system should react to external inputs like state changes or events from machines, operators, and sensors. Developing such systems can quickly become difficult due to all unforeseen situations one may end up in. Manual programming becomes too complex, and executing control sequences calculated off-line becomes very difficult to recover from when something goes wrong.

Several frameworks in the ROS community are helping users with composing and executing robot tasks (or algorithms). For example, frameworks like ROSPlan [19] use PDDL-based models for automated task planning and dispatching. There is also SkiROS [20] that simplifies planning with the use of a skill-based ontology. Moreover there are frameworks like eTaSL/eTC [21], which defines a constraint-based task specification language for both discrete and continuous control tasks, or CoSTAR [22] that uses Behavior Trees for defining the tasks. However, these frameworks are mainly focused on robotics rather than automation.

Based on what we *learned* and experienced, we have employed a combination of on-line automated planning and formal verification, in order to ease both modeling and control. Formal verification can help us create a correct model, while automated planning lets us avoid hard-coded sequences and "if-then-else" programming, while allowing us to be resource-agnostic on a higher level. We end up with a control scheme that continuously deliberates the best states to distribute to the devices. This is implemented in our research software Sequence Planner (SP), which is a tool for modeling, control, and analyzing of automation systems. A detailed explanation of the newest edition of SP, as well as control in intelligent automation systems can be found in [6].

2.4 Planning

The deliberative *core* of intelligent automation systems is a planner. Automated planning is a deliberative decision making process which yields sequences of actions that drive state change towards a goal [23]. It is a powerful tool in intelligent automation systems, as using planning in conjunction with acting, the control system is able to automatically adapt and re-plan based on the current state, virtually embedding the ability to restart.

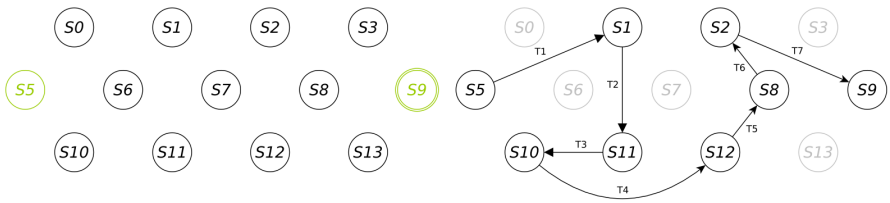


Figure 2.2: For example, given the current state of the system S5, the planner calculates a sequence of actions T1..7 that have to be executed in order for the system to reach the goal state S9.

A control system based on automated planning is goal oriented, meaning that after deciding a goal, it will calculate the necessary sequence of actions based on a model in order to reach that goal, Figure 2.2.

Example

Let's look at a well known PDDL [24] benchmark example, Blocksworld, as it is one of the most famous planning domains in artificial intelligence. Imagine a set of blocks sitting on a table. The goal is to build one or more vertical stacks of blocks. For each block in a specific problem, there are four actions that can be perofrmed: *pick*, *place*, *stack*, and *unstack*. The catch is that only one block may be moved at a time, and it may either be *placed* on the table or *stacked* atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved [25]. Such constraints can be defined as action guards, or as high-level specifications (invariants).

Based on an initial state and a goal state of a Blocksworld problem, as shown on the left and the right side of Figure 2.3 respectively, a planner can calculate the following plan: *pick_d*, *place_d*, *pick_b*, *stack_b*, *pick_a*, *stack_a*.

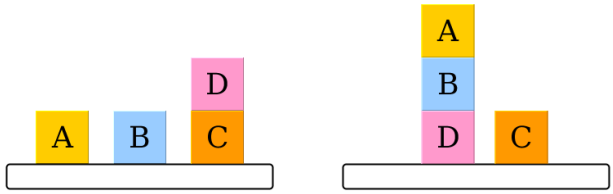


Figure 2.3: The Blocksworld example.

2.5 A possible run

When a pick order arrives, SP makes a *goal* from it saying that the items from the pick order should be on the autonomous platform. Next, a plan is calculated based on the current state of the system and this goal state. If nothing goes wrong, SP will execute the actions from this plan and finally reach the goal state. However, if something does go wrong, for example, a failed pick, SP will call the planner again in order to try to find a new plan that will take the system to the goal state from there.

If the robot has an attached tool, it has to move to the tool rack and leave it in order to attach the structured light scanner. After taking the scanner, the robot moves to above the blue bins in order to scan them for items. This time, the gantry has moved the base of the robot so that the box can be reached by the robot. The gantry mounts the robot and the tool rack, and is controlled by a PLC. In order to set reference positions, initiate movement, and read the current state of the gantry system, the PLC is connected to the ROS world using a ROS-OPCUA bridge [26].

After the scanning is done, localization, scene population and visualization of items is performed. The robot now goes up again to leave the scanner and to pick up the correct tool for one of the detected items. Around this time, SP calls in an autonomous platform so that it can eventually place the picked items on it. When an item is picked and the platform arrives, the robot places it after which it goes to change tool again in order to pick the second item. After both items were picked and placed, the platform goes out of the kitting station to the platform buffer. However, how does SP *know* about the state of the resources of the system, and how is data communicated in such a system? Let us study that in the next chapter.

CHAPTER 3

Investigating architectures

The performance of a system is tightly coupled to the control and communication architecture it *implements*. With *implementation*, we mean the type of control and communication architecture implemented within one resource and between several resources of an intelligent automation system. As it is described in [27], the development of robotics architectures is as old as the field of robotics itself. This applies for all control systems. So, in order to discuss different architectures and their applicability in different scenarios, let's try to identify and classify them first. We will first classify resource types based on their abilities and intelligence, and afterwards control and communication architectures based on degree of distribution and topology.

3.1 Resources

Resources are physical and digital assets which contribute to increased functionality of a system. They are represented by their corresponding models which group their local state and discrete descriptions of tasks they can perform [28].

We identify four different types of resources in intelligent automation systems:

1. *Stateless (projection type)*
2. *Stateful (procedural type)*
3. *Intelligent (deliberative type)*
4. *Operator (expert type)*

A resource is modeled with three types of variables, *command*, *measured* and *estimated*, where a *variable* is a named unit of data that can be assigned a value from its finite domain of values. Their valuation represents the *command*, *measured* and *estimated* state of a resource. The *command* and *measured* variables are also used to establish two way communication with a resource, while the *estimated*, or *memory* variables, keep track of state that can not be directly measured. Formal definitions of states, variables, and resources in intelligent automation systems can be found in [6].

Stateless resources

Stateless resources are simple devices and algorithms which don't own their own internal state. As such, they are not capable of *atomically* changing their state on their own, as it solely depends on external input. In other words, every atomic discrete state change of the resource has to be initiated by *someone*. Some examples of such resources are light indicators and simple grippers, however, more complex machines can also be controlled as stateless resources, for example robotic manipulators.

Models of stateless resources are usually a one-to-one representation of their capabilities, which means that the controller owns the complete and fine-grained state space of the resource and can thus have a pretty good idea of its actual state. Thus, the state of a stateless resource is a *projection* of the current *command* state of its controller. The concept of *stateless resources* is crucial for flexibility in automation systems, as the state can remain *centralized* in the controller itself, giving it complete control over the resource.

Example

Let's consider how it would be to control a robotic manipulator as a stateless resource. For this simple example, the robot can move between three poses, *a*, *b*, and *home*, so modeling the possible behavior for this robot yields 6 possible transitions: *a_to_b*, *b_to_a*, *a_to_home*, *home_to_a*, *home_to_b*, and *b_to_home*. Additionally, we can restrict the movement of the robot with certain high level specifications, saying that moving directly between poses *a* and *b* is forbidden, and as such, it always has to happen through the *home* pose. With this, we have removed the transitions *a_to_b* and *b_to_a* from the model, and we are left with 4 transitions, as indicated at the top of Figure 3.1 with *AH*, *HA*,*BH*, and *HB*.

The controller of this stateless resource owns the complete and fine-grained model, so it can initiate any possible atomic discrete state change of the resource. In Figure 3.1, the current state of the robot is in position *a*, and in order to fulfill a goal, which is for the robot to be in position *b*, a plan of two steps is calculated that will take the robot from *a* to *home*, and then from *home* to *b*. In order to do this, the controller has to be aware of all the possible states that the resource can be in, in order to initiate the correct actions in the correct state.

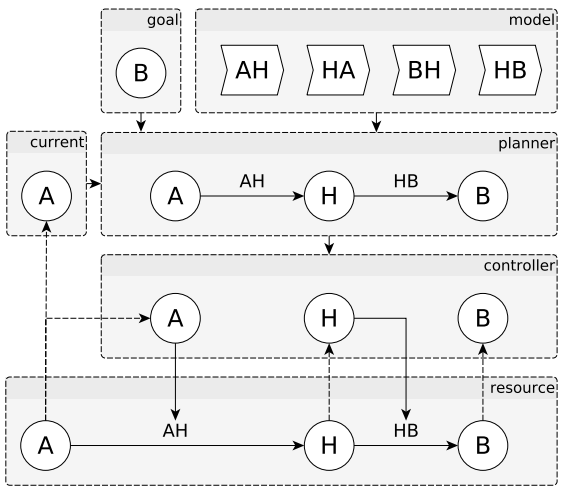


Figure 3.1: An example of how a stateless resource can be controlled.

Imagine now that the robot has encountered some kind of an unexpected error while moving between *home* and *b*, for instance a collision induced *protective stop*. Having access to such a fine-grained state space of the resource means that the controller, or operator, can choose how the error recovery should be performed. It is possible to put the robot in a *safe* state, which in this case can mean either *home*, *b*, or in some scenarios maybe even *a*.

Stateful resources

Not all resources can be controlled as stateless resources. Some resources require tight coupling with a real-time controller, or rather, they can only be controlled through a dedicated one. More often than not, this means explicit programming of control sequences, or *procedures*, which are initiated by an external controller. Once a procedure is initiated, a resource is able to move between its atomic discrete states independently, executing the actions of the procedure. As such, a part of the model is hidden from the high-level controller in the resource itself, and we refer to this as *decentralized* state.

Once a procedure is initiated, it can either successfully complete after some time, it can fail, or it can be aborted. Failing or aborting a procedure usually means a need to re-initiate the procedure from a safe state. Examples of stateful resources range from CNC machines, lathes, drills, 3D printers and robotic manipulators, to different kinds of detection, localization and navigation algorithms.

Example

Let's consider how it would be to control the same robotic manipulator as a stateful resource. When preparing and modeling the system, we might already know some information about how the robot movements can be performed. Instead of modeling the complete behavior and then restricting certain parts of it, like we did in the example for the stateless resource, we can choose to explicitly define two *procedures* that define the only possible way to move the robot, namely between *a* and *b* through *home*, and between *b* and *a* through *home*.

The controller now doesn't own the complete model, as it can only initiate high-level actions between *a* and *b*, i.e. it can not initiate all possible atomic discrete state change of the resource. In Figure 3.2, the current state of the

robot is in position a, and in order to fulfill a goal, which is for the robot to be in position b, a single step plan is calculated that should initiate the AB procedure to take the robot from a to b, while the procedure is doing all the necessary hidden steps.

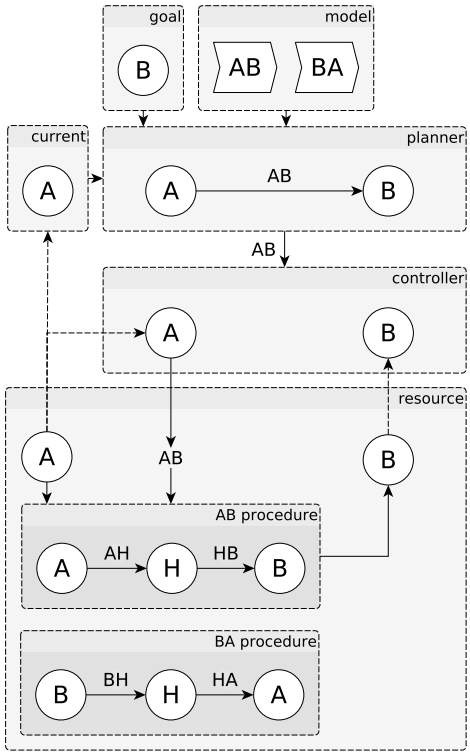


Figure 3.2: An example of how a stateful resource can be controlled.

Let's imagine again that the robot has entered the *protective stop* mode due to an unexpected collision between *home* and b. To recover from an error this time, the controller, or operator, can only choose between a or b for the *safe* state to put the robot in. If for some reason the movement between *home* and b has to be performed, the only option is to put the robot in a and re-initiate the procedure AB from the beginning. As such, there is a fundamental contrast in flexibility between stateless and stateful resources.

Example: Supporting handlers to manage state

In some cases though, there is good cause to move a part of the state out of the model, i.e. to *decentralize* the state, but not into the resources. For example, if there is a certain task that can be executed only within a strictly defined sequence, moving this behavior out of the model and into a *handler* node can help simplify preparation and modeling, simplify the resources, reduce the size of the model, and thus the strain on the planning and execution engines.

In the Rita use-case for example, there are several handlers that take care of behavior like changing tools, localizing items, and generating and executing robot control code. Let's look at these three examples, which are a good match to be handled outside of the model.

1. *Localizing items:* In order to localize items in the scene, several things have to happen. First, the scanner has to capture an image of the items, after which the localization algorithm has to identify those items and respond with some data. Such data has to be interpreted, manipulated and filtered so that only the best pickable items remain, after which the items have to be put into the scene by populating the *tf*. In the end, the detected and localized items have to be visualized. If any of these steps were to fail, the *localize_items* handler would respond with *false*.

2. *Robot code generation and execution:* Generating and executing robot control code also has a number of steps that have to happen in order. After, for example, a *move* command request has arrived, this handler node has to lookup several transforms from the *tf*, in order to know where is the *goal_frame* located in the robots *base* link, as well as where the *tcp_frame* is located in the robots *face_plate*. Using these frames, the robot is able to calculate an inverse kinematics solution in order to move to the goal frame with the required tcp frame. After generating this robot control code, it is sent to the driver so that it can be forwarded to the robot. This handler responds with *true* only if all of these steps were executed successfully and the robot motion was completed.

3. *Tool changing:* Since attaching and detaching different tools can only happen in one way, a handler node is made to execute such changing sequences. Tool changing sequences can be a bit long and hide a lot of details, however, we still chose to decentralize such behavior. After all,

if attaching a tool happened to fail, maybe the issue is of a mechanical nature. In this example, it can be observed that the tool changing handler node is a higher level handler which makes several calls to the robot code generation and execution handler.

There is no other possible sequence of actions that can perform these procedures, so we have to ask ourselves what are the trade-offs between having a possibility for fine-grained control of such a sequence, and having a simple procedure that can execute that sequence for us, while hiding a lot of the details. What we *learned* from Rita, is that it is OK to sometimes distribute state to *handler* nodes, if the tasks that they have to perform are strictly linear, meaning that there exists one way to execute the sequence, and in the case where it fails for some reason, it is reasonable to re-initiated it from the beginning. Another thing that we learned from Rita, is that there is no single way to solve all problems.

Intelligent resources

In some cases, it is appropriate to empower machines with deliberative abilities. It is easy to see the need for intelligence and autonomy in robots which are mission driven, for example autonomous drones, ships, submarines, and rovers. Quite often, mission driven robots have limited communication with their mission control base, which means that they can't rely on regular command updates. This demands a level of autonomy which can ensure that a robot can constantly sense an unknown environment, deliberate, and independently execute actions in order to fulfill a mission.

In an industrial setup, the environment can be much better observed and the communication can be made very reliable. However, that does not mean that there is no need for intelligent resources. For example, an intelligent resource is an autonomous transport robot (ATR) which delivers necessary assembly material to different stations. Given a schedule for the day, an ATR constantly deliberates in order to meet the deadlines imposed by the schedule, keep its battery charged and healthy, and plan its paths in order to avoid obstacles and save energy. Thus, an ATR's deliberative ability is probably not high compared to a rover on Mars, but it exists nonetheless.

As it was discussed, giving the resource a part of its model can sometimes help, particularly if the tasks it has to perform are easily *restarable* and strictly

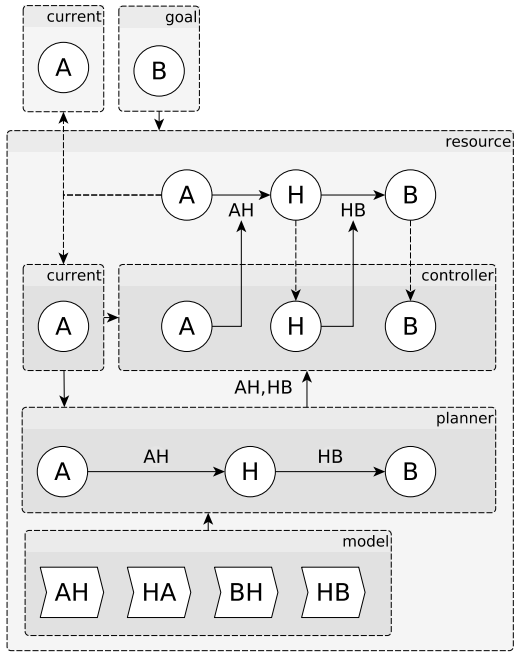


Figure 3.3: Overview of how a intelligent resource is controlled.

linear. However, the benefits of decentralizing state drop significantly as the tasks become more complex. Here are some of the reasons why we think that this is the case:

1. More complex tasks can often be executed in more than one way, and as such, programming procedures which execute them can be really hard and error prone.

2. Programming procedures for complex tasks goes back to traditional automation, as it removes all the flexibility that was a goal to be achieved in the first place.

3. Proper handshaking and state synchronization becomes extremely hard to achieve with the high-level controller, as complex tasks imply a lot of hidden state in the resource controllers.

4. Error recovery becomes extremely hard to achieve, as syncing the resources and controllers after an error to a state where an action can be triggered can be quite tedious.

An attempt to mitigate these drawbacks is not only to decentralize state, but intelligence as well. If resources can be equipped with intelligence, i.e. planning, execution, observation, and communication abilities, then having access to their state space could be a solution to decentralizing state, as well as reasoning and executing complex behavior. However, this doesn't solve all of the mentioned problems, as state that is decentralized is still hidden within resources. As a matter of fact, the multi-agent system community is trying to tackle this problem for quite some time now [29].

Multi-agent systems

In general, an *agent* is any entity that is capable of interacting with its environment. This interaction comes in the form of actions that can be motions, force exertions, perceptions, communications, etc. [23]. In multi-agent systems, *resource* and *product* agents are considered to be the main components. Additionally, these agents can have use-case specific specializations [30]. As there is a lot of different types of agents [31], other use-case specific agent types can also be included, hence a multi-agent system is not generally restricted to resource and product agents.

Agents in a distributed architecture are supposed to own internal state that is keeping track of the current process at hand. For example, product agents are supposed to keep track of the current state of the product they are bound to, and inform the other agents with which operations have to be performed on it in order to reach the goal state of the product agent.

Ideally, the actual state of the product should be known by the product agent by doing constant measurements. However, this is usually not the case, since in some cases it is inefficient to do so, and in some cases even impossible. Because of this, product agents have to *estimate* the current state of the actual product by updating their internal state based on the operations performed by the resource agents. This makes handshaking between product and resource agents a bit tricky, as resource agents also own internal and *estimated* state in order to keep track of performed operations.

Having estimated state is usually quite helpful and sometimes unavoidable, however keeping track of the same estimated state in different places can cause

a lot of synchronization issues. In order to keep the state between product and resource agents synchronized, multi-agent systems usually either depend on real-time communication, or on some sort of *watchdog* agents [32] which purpose is to keep track of states of other agents and trigger specific actions if the agents get out of sync.

Isolation

An agent performing actions deliberately is motivated by some intended objective. As agents may have only a partial model of their environment, this objective is usually formed by interacting with other agents [33]. This interaction between agents exhibits emergent behavior that characterizes multi-agent systems.

While admiring the core idea of multi-agent systems, our view is that such behavior is extremely hard to achieve and control. Instead, we see intelligent resources as components of automation systems that can perform virtually *isolated* tasks. Looking at autonomous platforms, submarines, drones, and rovers, it is not hard to see that such intelligent resources with high deliberative skills don't have to constantly interact with other resources in order to achieve a coordinated collective task. If a collective task were to be achieved, it would probably be easier to employ a high-level coordinator, or some kind of a hierarchical control structure.

Example

In the Rita use-case, we did not employ any intelligent resources as they are defined here. However, one job seems suitable and might be a good test case for future experimentation, namely discrete robot motion planning and execution.

As there is a lot of frames defined in the scenario, there is also many ways a robot can move between them. For a number of frames in the scenario $n = 60$, and the number of tcp's the robot can move with $m = 7$, the number of possible motions is:

$$\frac{n(n-1)}{2}m = 12390$$

Of course, not all movement should be enabled in order to avoid collision, so the real number of allowed moves is smaller. Nevertheless, all the positions, via points, parameters, and constraints for moving the robot are currently part of the monolithic model for this scenario, which takes up a big chunk of it. Moreover, if there were many other frames, which is not hard to imagine, it would be harder and more time consuming to take care of all robot movement specific behavior in a monolithic model.

Instead, it could be viable to implement an intelligent robot motion planning and execution resource, which owns the robot movement model, receives goals from the high level controller, calculates correct motion plans, and executes them. As seen on Figure 3.3, the model is owned by the resource, and when it receives a goal, namely to move to position B, the resource plans and executes a correct motion.

Operators as resources

Human operators have *expert* knowledge about the system and the process at hand. As such, they can perform necessary operations as instructed or self-initiated, both quickly and efficiently. Operators can recover the system from faults, adapt their pace to reach a deadline, and move between different stations to balance out the workload.

In intelligent automation systems, there has to be a way to exchange data with operators. Operators can exhibit properties of all previously defined resource types, and as such, it is the data exchange *agreement* between the control system and the operators that defines the role of the operator in an intelligent automation system.

Example

In the Rita use-case, the robot is picking requested items from an order and placing them on the moving platform. If, for some reason, the robot is unable to pick some of the items from the order, a *pick-to-light* system informs operators that such items have to be picked manually. It is a two way method of communication, as the lights inform the operator which items to pick, after which the operator informs the system that the item was picked by pressing the lit button. In this scenario, the operator and the system *agreed* to exchange data for picking every individual item, and as such, the operator

exhibits properties of a *stateless* resource.

In some cases, some pre-assembly has to be performed by the operator using the picked items. If the station is properly equipped, the assembly command together with the assembly instructions can be shown on one of the screens in the station. After the pre-assembly has been finished, the operator has to inform the system somehow. This kind of *agreement* with the system makes the operator resemble *stateful* resource properties.

As the operators possess *expert* knowledge of the system, they should be able to perform necessary tasks with or without instruction. Keeping track of the station's health, moving the material facade, loading full and unloading empty boxes and crates, recovering from errors, etc., shows the true power of intelligent automation systems that include operators.

For example, in the Rita use-case, adding a scanning frame for a new item type can be performed by operators. Instead of stopping production for a long time during which specialized robot programmers have to be called in to program a new pose, an operator that is acquainted with the system should be able to do the same thing, online. Changing the system to *teaching* mode gives the operator permission to manually guide the robot to a desired pose. As the robot is equipped with force sensors, it conforms to the guidance forces exerted by the operator. Moreover, the operator can always know what the robot sees, by streaming the first person view of the scanner mounted on the robot to a screen in the station.

Safety

Operators and machines can interact in several different ways inside an intelligent automation system [34]. Probably the most challenging way of interaction is collaborative, where humans and robots directly interact in order to complete a common complex task. As shown on Figure 3.4, a dedicated 3D safety system can be used to keep track of the proximity between the robots and humans. In order to keep interactions safe, such proximity data can be used to employ some of several collaborative operation modes that have been defined over the years [35]. However, it seems like such modes, at least in pure form, are too restricting in order to achieve truly flexible and efficient human-robot collaboration.

Since operators have *expert* knowledge about the system, the nature of movements and operations performed by the robots are well known. As such,

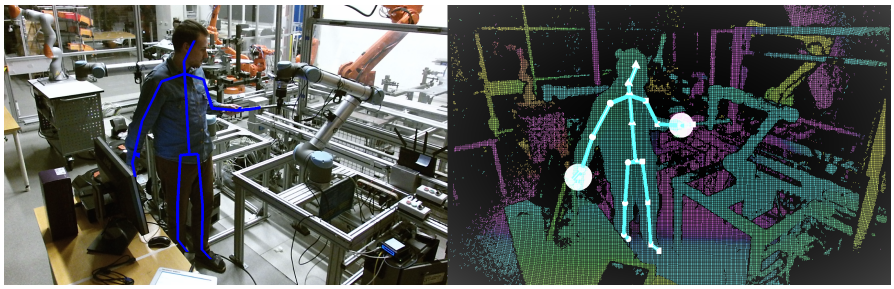


Figure 3.4: Example of a dedicated operator tracking system.

it might be possible to employ a new type of *deliberative agreement* [36], where the operators behave within the agreed limits of deliberative collaboration in order to ensure safety.

3.2 Communication

We distinguish two kinds of communication models in intelligent automation systems. Choosing one or the other depends on the type of the resource that is communicating, the type of data that is communicated, and the type of task the resource has to execute. We identify these kinds as:

1. *Event based communication*

2. *State based communication*

Event based

In an *event based communication* model, two nodes are exchanging data in an irregular time pattern. When *something* has to happen, data is sent over the network by a node in a form of an *event*, which triggers certain action in another node that is receiving that event. This means that event identifiers have to be mapped to certain actions in the receiving node. If the event receiving resource is stateless, it will execute *atomic* discrete actions mapped to event identifiers. On the other hand, if the event receiving node is stateful, it will execute certain *procedures* mapped to event identifiers, which might

not be atomic. In intelligent resources, event identifiers could be mapped to certain goals that the receiving node should try to reach.

Example

In the Rita use-case, an event is sent to the robot controller node in order to trigger a robot motion. Events can easily be parameterized to carry more data and provide additional value. A parameterized event is represented by a message type on the left side, while a message instance can be seen on the right side:

```
string command                              move_j
float64 acc                                 0.3
float64 vel                                 0.5
bool use_execution_time                     false
float32 execution_time                      0.0
bool use_blend_radius                       false
float32 blend_radius                        0.0
bool use_joint_positions                    false
JointPositions joint_positions              default
bool use_preferred_joint_config             false
JointPositions preferred_joint_config       default
bool use_payload                            true
float32 payload                             1.375
string goal_feature_name                    above_box_3
string tcp_name                             svt_tcp
int32 tf_lookup_deadline                    3000
```

After the robot controller node has performed all the necessary actions, it can respond with an event to the first node telling it that the actions were completed successfully, or that they have maybe failed:

```
bool success                                true
```

As it is probably hinted at, an event based communication model is suitable for communicating commands with stateful and intelligent resources. As these nodes already own state, using events as triggers for their internal state change seems like a natural choice.

State based

In a *state based communication* model, two nodes can exchange data either in a regular or irregular time pattern. If the pattern is regular, it can be looked at as data *streaming*. The data sent between nodes is now not in a form of an event, but state, which can directly be written to the state of the receiving node. This communication model is useful for sending state based commands to *stateless* nodes, as well as streaming certain state, for example *joint_state* of a robot. Moreover, *stateful* and *intelligent* resources can use this communication model to stream information about their internal state, for example data about some readings, or about the current step of the procedure the node is executing.

Implementation

In ROS based intelligent automation systems, as defined in this thesis, event and state based communication models can be implemented using the following tools:

1. *Topics* are data channels on which information is streamed. Nodes can *publish* data to topics at a certain rate, while other nodes can *subscribe* to them so that they can receive data. Publishing is useful for sending command state to stateless nodes, streaming local state or streaming data from sensors and cameras.

2. *Services* are procedures that can be initiated by clients using a request - reply type of interaction. A server offers a *service* which it can execute on a service topic, so that a client can send requests on that topic to the server. We *learned* that services are useful for *event based communication* where the actual tasks take very little time to execute, or where it is certain that the server can execute the tasks.

3. *Actions* are also procedures that can be initiated by clients using a request - reply type of interaction. In contrast to regular services, action clients are not blocked while awaiting a response. Moreover, an action server can simultaneously publish the execution state of the action, and it allows the action to be aborted or paused. Actions are suitable for *event based communication* where the actual tasks take a long time to execute, or where it is not certain that the server can execute the tasks.

3.3 Control

Control architectures define the coupling and interaction of controllers and the system that is controlled. The two main classes of control architectures that are well known are centralized and decentralized control systems [37].

Based on the types, location, amount, and behavior of processes that should be controlled in a system, engineers can choose to either implement centralized or distributed control. There are pros and cons in both types of implementation. For example, large scale distributed control systems should be relatively easier to maintain and update from large scale centralized control systems. On the other hand, it is sometimes impractical or inefficient to decentralize control, especially when controlled processes are tightly integrated [38].

There is more than one way to decentralize control based on the data that is communicated between the controllers, and the existence of hierarchies or higher level supervisors [37]. As such, we identify the following control architectures in intelligent automation systems:

1. *Centralized*
2. *Distributed*
3. *Hierarchical*

Centralized

Centralized control architectures imply a single point of control, i.e. a single controller, where the sensors and actuators can be distributed in the field. Such control system architectures are suitable for controlling smaller systems, or systems where the complete control logic has to be run from one machine. In mission critical systems, a *redundant* controller should be able to execute the same control behavior in case of a fault in the main controller.

The benefits of centralized control architectures are the direct interaction between controllers and resources, a relatively simpler communication infrastructure, and ease of implementation. The drawbacks include the difficulty to scale, and without considering redundancy, a single point of failure. Thankfully, tools like *docker* and *kubernetes* can be utilized to automatically deploy containerized controller and resource nodes for better scalability, or to recover after a fault in a cloud-like setup.

Distributed

Distributed control architectures consider two or more distributed and *autonomous* controllers without any hierarchies or higher level supervisory control. This architecture enables distribution of controllers in the field, which in turn enables a tighter coupling with their corresponding field sensors and actuators. As a rule of thumb, the more physically dispersed resources are, and the more operationally independent of each other their processes are, the more sense it makes to distribute control [38].

The control agreement and the data that is sent between controllers in a distributed setup can produce different kinds of behavior. An example could be sending parameters between two controllers on an assembly line, where the former updates the parameters of the latter based on the current product that is manufactured. Even if there is communication between these controllers, there is no hierarchy between them as they execute control code independently. Advantages of distributed control architectures are modularity, scalability, and easier maintenance, while disadvantages include relatively more complex communication, as well as implementation.

Hierarchical

A system with a *hierarchical control architecture* consists of two or more layers of controllers organized in a hierarchical topology. At the top of the hierarchy, there is always a main controller which generates control for the lower levels. The control is then *refined* through the levels until it eventually reaches the resources.

In contrast to distributed control architectures, the data exchange between controllers in a hierarchical architecture ensures that the lower level controllers always behave within the control rules of their higher level controllers.

The Rita use-case employs a hierarchical control architecture, where Sequence Planner deliberates control state, commands, and goals for stateless, stateful, and intelligent resources respectively. For instance, a multi-level hierarchy is quite visible in the tool change example, where SP issues a tool change command to the tool change handler, which in turn issues commands to the high-level robot code generation and execution handler, which then sends individual move commands to the robot driver.

Honorable mentions

In Supervisory Control And Data Acquisition systems (SCADA) [39], autonomous controllers are distributed in the field while having access to a shared database. While these controllers mostly act autonomously, a high level supervisory system can modify parameters in the shared database, practically forcing the field controllers to act within the bounds set by the supervisor.

Another example are multi-agent systems, which can exhibit different behavior based on the architecture they employ. There is a number of architectures defined for multi-agent systems [40], and as such, multi-agent systems can have a very flat hierarchy, a very strict hierarchy, as well as everything in between.

CHAPTER 4

Supporting infrastructure

Intelligent automation might be the answer to some of the production challenges mentioned in Chapter 1. However, being a novel approach, it lacks supporting tools and methods which mature and reliable technologies nowadays take for granted. Software for design, simulation, commissioning, control and maintenance of intelligent automation systems either doesn't exist or it is in infant stage of development. For example, industry leading software tools for traditional automation systems like Process Simulate and Delmia include support for simulation and virtual commissioning [7]. However, neither of those solutions is applicable to intelligent automation systems, nor does other software exist which can support development, simulation and virtual commissioning of such systems.

4.1 Virtual preparation

For virtual preparation and control of systems, we define a notion of *scenarios*. A scenario is represented by its resources, their arrangement and the task that they are going to execute. Preparing such scenarios is one of the first steps of realizing an intelligent automation system. In order to do that efficiently,

we heavily utilize *tf*, which is a ROS package that lets the user keep track of multiple coordinate frames over time. It does so by maintaining relationships between coordinate frames in a tree structure buffered in time, and lets the user manipulate transforms between any two coordinate frames at any desired point in time.

We like to look at *tf* as having two ways of getting information to its tree and maintaining coordinate frames:

1. *static transform broadcasters*

2. *active transform broadcasters*

Static transform broadcasters publish transforms that are static in time, i.e. the transformation between the frame and its parent is not expected to change over time. Because of this, the time dimension can be disregarded and the transforms can be published very sparsely. Examples of static transforms could be bolt holes defined in the engine frame, or a sensor defined in the camera's mounting point. These transformations define where such frames are, relative to their parent frame, and they are not expected to ever change.

Active transform broadcasters publish transforms that are active in time, i.e. the transformation between the frame and its parent is expected to change over time. We now have to publish the transform together with a timestamp, so that *tf* can keep track of changing transformations in time and provide us with the correct answer when we look up the *tf* tree at a certain time point. Examples of active transforms range across mobile robot bases defined in a world coordinate frame, products that are transported on a conveyor belt, and multiple DoF manipulator links defined in their parents links via joints.

Example

Let's look at the Rita use-case and how we would prepare a scenario for picking some items and placing them on a moving platform. We start with supplied original CAD files of components, and modify their coordinate frames having in mind how they should couple or interact with other components of the system. For example, a tool for picking certain items should have a *tool center point* frame at a correct place, as well as a *main* coupling frame. The left side of Figure 4.1 shows how the main frame of a suction tool was set, in order to properly couple with the robots *rsp* connector. The right side

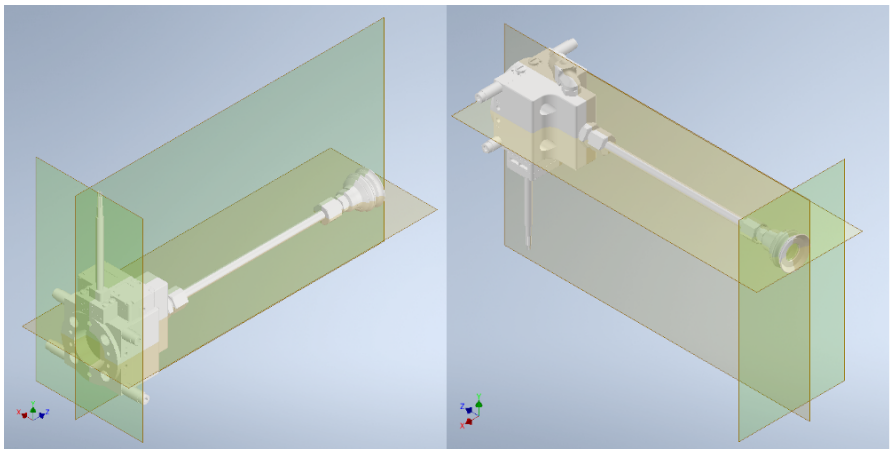


Figure 4.1: Preparing tool frames in Inventor.

of Figure 4.1 shows how the *tool center point* was set in order to match the frames of items to be picked up with this tool.

Such frames have unique names in order for *tf* to be able to keep track of them. Let's call the main frame *svt*, short for *small vacuum tool,* and the tool center point frame *svt_tcp*. The *svt_tcp* is defined in the main *svt* frame as a static transform since it is not expected for the transformation between them to change. On the other hand, the main *svt* frame is expected to be picked up by the robot or left at the tool stand, so let's define the *svt* frame as an active transform with an initial parent at the tool stand. As the tool is expected to have a *home* position when released by the robot, let's call this frame *svt_home*.

To define these relationships, our current scenario has a collection of frame description files in a *.json* format. Let's look at the descriptions of these two frames:

svt.json:

```
{
    "show": true,
    "active": true,
    "child_frame": "svt",
    "parent_frame": "svt_home",
    "transform": {
        "translation": {
            "x": 0.0,
            "y": 0.0,
            "z": 0.0
        },
        "rotation": {
            "x": 0.0,
            "y": 0.0,
            "z": 0.0,
            "w": 1.0
        }
    }
}
```

svt_tcp.json:

```
{
    "show": true,
    "active": false,
    "child_frame": "svt_tcp",
    "parent_frame": "svt",
    "transform": {
        "translation": {
            "x": 0.0,
            "y": 0.0,
            "z": 0.313365
        },
        "rotation": {
            "x": 1.0,
            "y": 0.0,
            "z": 0.0,
            "w": 0.0
        }
    }
}
```

The measurements in these description files are taken directly from Inventor when preparing the tool frames. As such, if the CAD file exactly matches the real tool, we can be quite confident that the *tf* tree will be an accurate representation of the real world.

As we have prepared the tool frames, we can use the same technique to prepare everything else in the scenario, including the items that have to be picked by the robot. Additionally, extra frames can be added to pickable items, since their orientation in the box they arrive in is random, and it could be that their main frame is occluded or not accessible. For example, let's look at these two items on Figure 4.2.

If these items would be placed in a grid with a nice accessible orientation, there would be no need to add extra frames to them. However, if an item is turned around, there is no way to pick it. In order to mitigate that, we add

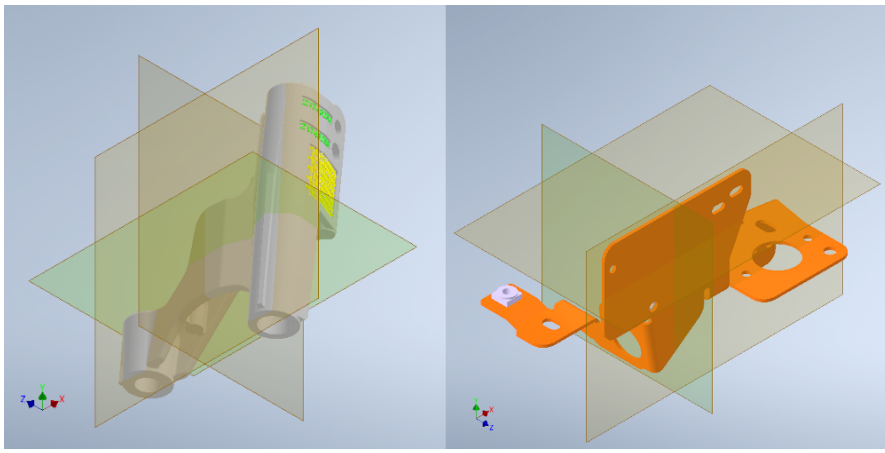


Figure 4.2: Preparing item frames in Inventor.

additional transforms to the item's description file which are defined in the main frame of that item. For example, for the item on the left side of Figure 4.2, we can add the following secondary picking points, and we can do this directly in *rviz*, using a modified version of the *interactive_markers* package as seen on Figure 4.3.

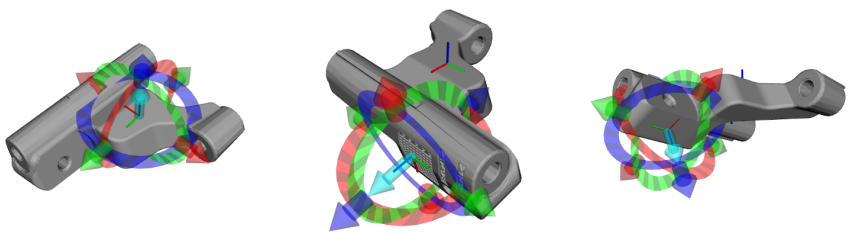


Figure 4.3: Preparing secondary item frames in Rviz.

Another possibility would be to automatically compute possible grasping points based on some kind of surface matching and area comparison algorithm. What we get from this step of the preparation are frames that define secondary picking points for items.

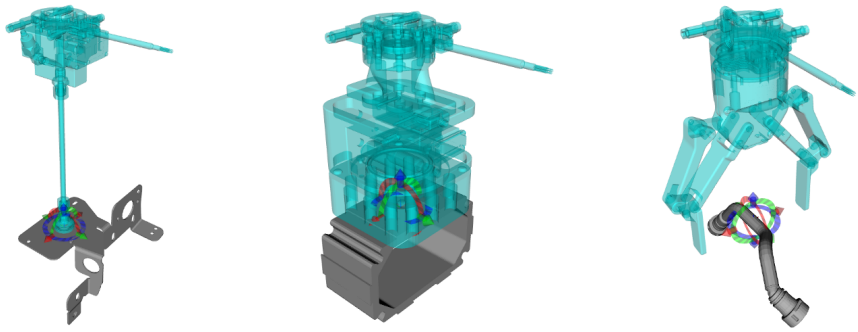


Figure 4.4: Verifying item frames with tools in Rviz.

Finally, it is possible to verify the picking points and determine which tool is to be used for picking which item, as shown in Figure 4.4.

After preparing the scenario, we end up with a lot of description files for frames that are in that scenario. A detail of a prepared scenario is shown in Figure 4.5.

4.2 Emulators and code generation

Modeling behavior of a system can be quite hard, as a lot of details can be easily overlooked. To use resource transitions in planning algorithms and to support model testing without being connected to the simulation or the real environment, *measured* variables must be updated by *someone*.

In an SP model, it is the *effects* that define how measured state variables change during execution. However, instead of letting Sequence Planner perform the updating of measured variables internally, we have chosen to place this updating as individual *emulation* nodes [28].

Contrary to simulations, the internal state of an emulation does not have to reflect the internal state of the target which it is emulating, it is enough for it to mimic the interfacing behavior of a real target. Because of this, the internal structure of emulator nodes is quite simple and can be automatically generated based on the model. Using emulator nodes has the following benefits:

1. Allows for more realistic testing during modeling, as the updating is not done internally, i.e. the emulator nodes have to interact with Sequence

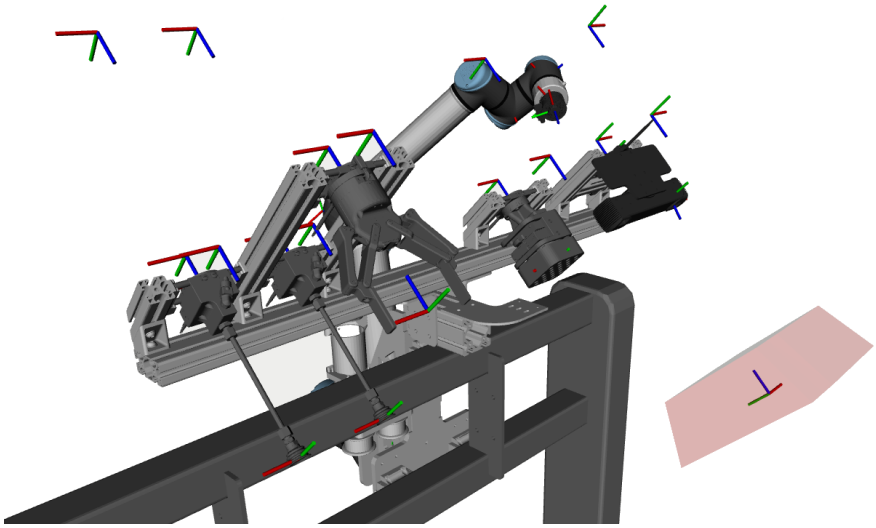


Figure 4.5: Preparation detail of the tool stand.

Planner through the, also autogenerated, messages on the network.

2. The structure of the network is partially generated and maintained during modeling, which allows for an easy transition to simulated or real environment.

3. The core execution engine is kept uncluttered, as the updating of measured variables doesn't have to be taken care of. It is happening outside, in the emulator nodes.

Example

Let's model behavior for the gripper used in the Rita project, which can be seen on the right of Figure 4.4. Variables and their domains that we are going to use in the initial model are:

gripper_state = {opened, closed}
gripper_command = {open, close}

Let's write a few simple transitions that open and close this gripper:

name	guard	update	type
close_start	*opened* ∧ *open*	*close*	*control*
close_finish	*opened* ∧ *close*	*closed*	*effect*
open_start	*closed* ∧ *close*	*open*	*control*
open_finish	*closed* ∧ *open*	*opened*	*effect*

Table 4.1: The initially modelled transitions of the gripper resource.

From Table 4.1, we can see that *gripper_state* variable has to be updated by someone when the *effect* transitions happen, so the gripper emulator node is generated to do the updating for us.

4.3 Virtual commissioning

Simulation of production systems is a well adopted practice. Industry leading software tools like Process Simulate and Delmia include support for traditional virtual commissioning (VC) [41], which exposes the simulation model to a real control system. Using these tools for designing control systems and performing VC has become a standard in industry.

The purpose of VC is to enable the control software, which controls and coordinates different devices in a production station, to be tested and validated before physical commissioning. Over the years, the VC community has specified several commissioning configurations [42], [41] that resulted in terms like *hardware-in-the-loop*, *reality-in-the-loop* and *constructive* commissioning. As shown in Table 4.2, these specifications are defined by combinations of components being real or virtual.

Testing and integrating the physical production system with the real control system has been traditionally referred to as physical commissioning. However, in order to reduce the amount of on site man-hours during physical commissioning [43], the real control system is coupled with a simulation model of the production system creating a hardware-in-the-loop setup. This configuration is commonly known as VC [41].

Performing the inverse of VC can be beneficial in situations when debugging

plant	controller	type
real	*real*	*physical*
real	*virtual*	*reality-in-the-loop*
virtual	*real*	*hardware-in-the-loop*
virtual	*virtual*	*constructive*

Table 4.2: Traditional commissioning classification.

the control system is needed. In this case, the physical production system is controlled by a simulated controller in a setup known as reality-in-the-loop. When designing a new control system, the natural way is to start with offline programming where all components are simulated. This configuration is also known as constructive commissioning [41].

Performing VC can reduce testing and integration time, as well as help detect undesired behavior before physical commissioning. However, it is usually the case that creating simulation models requires extensive modelling effort. Because of the cost associated with this effort, it is crucial that the created models provide as much additional value as possible.

A knowledge gap exists between simulation and control system development since they are traditionally decoupled activities in industry [44]. This sparked the emergence of a new concept, *integrated virtual commissioning*, which aims to integrate VC into the standard engineering workflow as a continuous support for automation engineers [45], [46]. Instead of using VC as the last step before physical commissioning, integrated VC enables simulation supported production preparation and automation engineering by simultaneous development of the control system and the virtual plant [46].

Example: Virtual commissioning

When developing an intelligent automation system, the first step is usually to develop a simulation of that system. This means that the main resources of that system have a simulated version, and a driver that is adapted to control such a simulation. Ideally, this driver should capture as much behavior as possible so that it can later be used to control the real resource without the need to change it much.

In order to verify the behavior of simulated resources using their accompanying drivers, *dummy* nodes are made which are quite useful to test some targeted behavior during development. These *dummy* nodes emulate the controller by providing some dummy input to the driver nodes for which their behavior is tested. This step can be looked at as *constructive commissioning* since both the resources and the controllers are simulated.

After the individual simulated resources behave as intended, the next step is to use the initial model that was developed with the help of emulators, in order to test interactions between resources. Writing dummy nodes to test these interactions can be quite tedious. Instead, the real controller is plugged in, so this step can be looked at as *hardware-in-the-loop commissioning.* There are three things developed in this step:

1. *The model:* Modeling behavior for a system is quite a challenging task. Using generated emulators to develop an initial model is helpful, however it is usually the case that a lot of behavior is unaccounted for, so the only way to generate a more or less complete model is to iteratively develop it while testing certain parts of it using a simulation. For instance, continuing with the *gripper* example, using the initial model with the gripper simulator reveals crucial behavior that was previously missed. Namely, the gripper takes certain time to open and close, and as such, the initial domain of the *gripper_state* variable is not enough to represent what is actually happening in the simulator. Thus, let's add another possible value that this variable can have while the gripper is opening or closing.

name	guard	update	type
close_start	$opened \wedge open$	*close*	*control*
close_finish	$(opened \vee unknown) \wedge close$	*closed*	*effect*
open_start	$closed \wedge close$	*open*	*control*
open_finish	$(closed \vee unknown) \wedge open$	*opened*	*effect*

Table 4.3: The updated transitions of the gripper resource.

2. The controller: As mentioned earlier, intelligent systems can't depend on existing traditional control solutions as the requirements make it impossible to use them. Controllers for intelligent systems contain optimizers, planners and execution engines which also have to be tested for correctness and eventually adapted for the current use case.

3. The resources: A lot of important resource level behavior can be overlooked while developing drivers without a model that tests their interaction. That is why it is usually the case that resource drivers are re-visited in this step.

The time has come to implement the real system. During setup, we might want to test some very special and targeted behavior, for instance something like a single movement of a robot or taking and processing a single snapshot. Now the *dummy* nodes are utilized again, this time emulating control for real resources. This step is analogous to *reality-in-the-loop commissioning*. Switching to a real system from a simulation always means discovering a lot of hidden behavior, hence a lot of changes have to be implemented in the drivers as well in the dummy nodes in order to properly test them. For instance, let's look at the gripper example again. Important information that the actual gripper is able to provide that was missed during the emulation and simulation steps, is that the gripper can not only be open or closed, but also gripping an item. This can later be modelled as an additional *effect* transition, since closing the gripper can either result in completely closing it, or actually gripping something. We also have to update the domain of the *gripper_state* variable to be able to capture this.

name	guard	update	type
close_start	$opened \wedge open$	*close*	*ctrl*
close_finish_c	$(opened \vee unknown) \wedge close$	*closed*	*eff*
close_finish_g	$(opened \vee unknown) \wedge close$	*gripping*	*eff*
open_start	$(closed \vee gripping) \wedge close$	*open*	*ctrl*
open_finish	$(closed \vee unknown \vee gripping) \wedge open$	*opened*	*eff*

Table 4.4: The once again updated transitions of the gripper resource.

Finally, it is time for *physical commissioning*, which means controlling the real system with a controller based on the model that was developed in the earlier steps. Once again, a lot of unaccounted behavior can arise, which has to be addressed by modifying the model, drivers, and sometimes even the controller.

4.4 A digital twin

Current definitions of digital twins have different shapes and sizes [47]. One definition explains digital twins as virtual representation of a physical system, where data flows between a physical system and its digital twin are fully integrated in both directions.

For us, this is probably not the case, however for a crucial part it is, namely tracking positions of items, tools, robots, etc. Moreover, in order to ensure safe human-robot collaboration, and maybe even deliberative collaboration, a digital twin can be used to visualize the state of the system on a screen. This way, an operator can know about the state and intentions of the system.

Keeping track of where things are in a reliable and continuous way is of utmost importance for realizing truly flexible automation systems. In order to do this, we again heavily utilize *tf*, the ROS package we relied on to prepare our scenario.

Example

In a flexible automation system like Rita, nothing (except for a few cameras in the ceiling) has a fixed position. Automated platforms that carry the picked items, material facades that hold the boxes with items, and even the gantry that mounts the robot and holds all the tools can freely move around in the station. Moreover, the items that have to be picked from the boxes arrive in a arbitrary order. So, how can we know and track where things are?

For large items in the scene, like the platforms, different crates, the robot gantry, and material facades, we use a set of *aruco* tags [48]. Detecting these tags and estimating their position is part of the task, however these positions have to be moved into the *tf* tree in order to *know* where they actually are in relation to something else.

Figure 4.7 shows how the tracking of the material facades is performed. A *live* position of the material facade is continuously updated in the *tf*. After

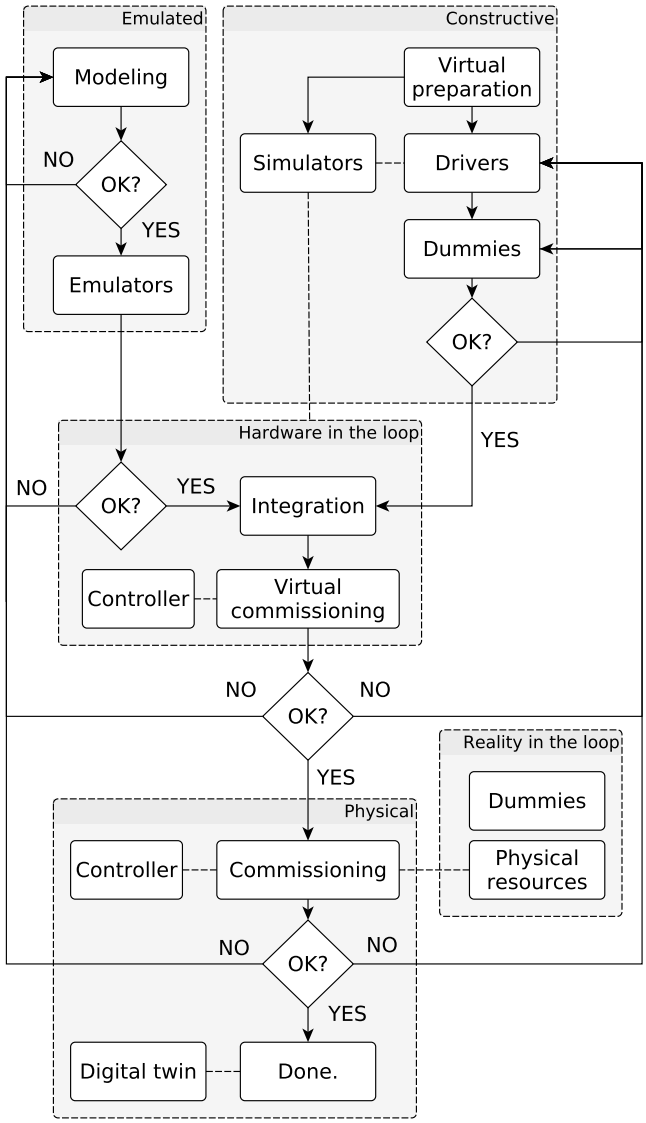


Figure 4.6: Preparation and commissioning workflow.

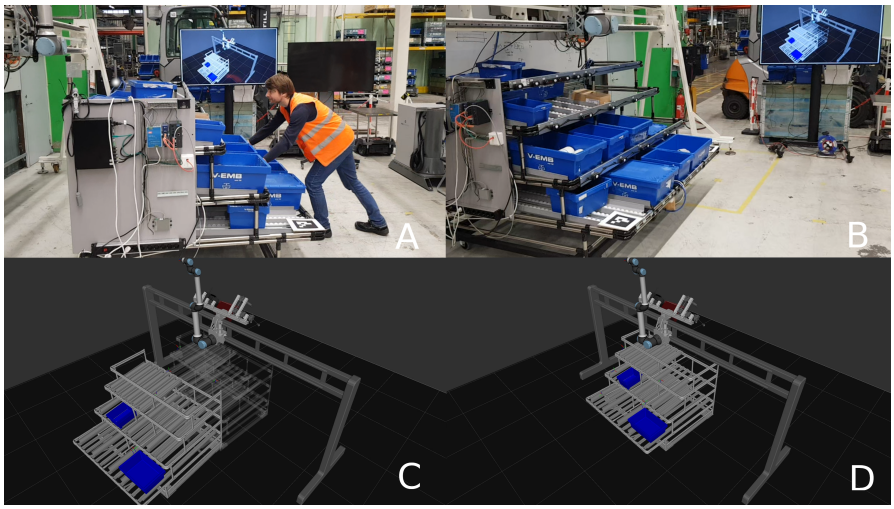


Figure 4.7: Tracking and locking the position of the material facade.

the operator has pushed the material facade under the gantry, as shown in parts A and B of Figure 4.7, the faded *live* position of the facade in part C is immediately updated in the *tf*. However, small detection distortions can make the real-time position estimation of the *live* facade frame quite unstable. If we would to define the boxes and items as children of this unstable frame, we would risk potential robot collisions. Instead, we generate an additional *locked* facade frame as a direct child of the common parent of the gantry frame when the operator is satisfied with the actual facade placement. Locking the frame is shown as a transition between parts C and D of Figure 4.7.

It is *tf*, or in this context, the *digital twin* of coordinates that keeps track of where the facade is, and thus also the boxes that hold the items. If the robot has to move to a position where it can scan a box for items, it has too ask the *tf* for that position. Only then can the robot use that position to calculate a motion to the box.

When scanning the items, the positions of detected items are also added into the *tf* tree, as children of the box frame where the scanning was done. Then, the robot can move with, for instance, the *svt_tcp* frame to one of the item frames that was earlier prepared in order to pick it. The scanning and populating the *tf* for two item types is shown in Figure 4.8.

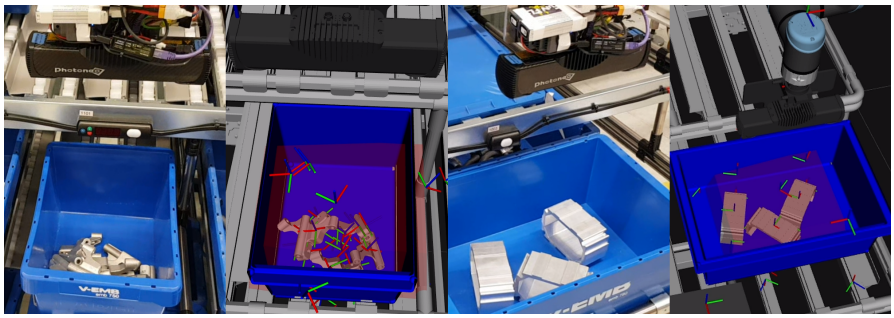


Figure 4.8: Scanning items and tracking their positions in tf.

4.5 Model based testing

Unit-testing is a natural part of modern software development. In addition to the emulator nodes generated from SP models, *test* nodes can also be generated. Since the SP model clearly specifies the intended behavior of the nodes via *effects*, it is also possible to generate tests which can be used to determine if the model and the underlying driver implementation do in fact share the same behavior. The generated tests are run over the ROS network, using the same interfaces as the control system implementation. This enables *model based testing* using a seamless mixture of emulated, simulated, and real device drivers, which can be configured depending on what is tested. Based on what we *learned*, the generated unit tests can:

- ensure that the device drivers and simulated devices adhere to the behavior specified in the SP model.
- help eliminate "simple" programming errors that waste development resources.
- provide means to validate if the specifications in the SP model make sense. While formal methods can guarantee correctness w.r.t. some specification, writing the specification is still difficult and needs to be supported by continuous testing.

The generated tests employ property-based testing using the Hypothesis library [49], which builds on ideas from the original QuickCheck [50] implementation. These tools generate random test vectors to probe if they can

break user-specified properties. If so, they automatically try to find a minimal length example that violates the given properties. The properties that need to hold in our case are that effects specified in the SP model always need to be fulfilled by the nodes implementing the resource drivers. Of course, when bombarded by arbitrary messages, it is not surprising to also find other errors (for example, crashes). As such, *test* nodes can be looked at as advanced *dummy* nodes, which do not only test a resource for a specific behavior, but for all possible behavior specified by the test.

CHAPTER 5

Investigating intelligence

Intelligent automation solutions are slowly but surely making their claim in automating production processes. However, naming something *intelligent* has different meaning in different research communities, so let's define what intelligence means in the scope of this thesis.

Intelligence has different aspects, ranging from learning, memory and knowledge, all the way to planning, creativity and decision making. For example, [8] defines intelligence as a very general mental capability that involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly, as well as learn from experience. For us, intelligence is the ability to reason about a specific state and try to make decisions in order to solve a problem.

We use this intelligence mainly as a tool to plan and deliberatively act towards a goal while avoiding undesired behavior [51], but also as support for modelling, testing and commissioning automation systems [52]. To achieve this, we use SAT solvers [53] to calculate and verify assignments of state variables, which enables us to use the same tooling developed around a solver while being able to encode and solve different kinds of problems.

5.1 Planning

Algorithms that compute plans based on explicit state-space searches exist since the late '50s [54], and symbolic methods based on BDDs since the late '70s [55]. A more recent method that has established itself as an important planning approach is SAT-based planning. Planning problems are encoded as satisfiability problems and the results are calculated by SAT solvers.

Even though SAT-based planning was first proposed by Kautz and Selman already in 1992 [56], the interest of planning researchers in SAT-based planning methods was limited up until relatively recently. One of the main reasons behind this was the performance advantage of explicit state-space search over solving early SAT encodings of planning problems [57]. However, modern planners based on satisfiability now match, and often outperform, planners based on other search paradigms [58].

Explicit state-space search and symbolic methods based on BDDs are known for their performance in solving problems with a small number of state variables, however SAT-based methods excel in solving hard combinatorial planning problems with a relatively high numbers of state variables [59]. Another advantage of planners based on SAT is that algorithms used to compute plans are almost completely general purpose SAT solving algorithms, which means that every improvement in the solver directly improves planning.

5.2 Solvers

SAT-based planning relies on SAT solvers to calculate satisfiable assignments for a Boolean expression that encodes a plan. A Boolean expression is satisfiable if there exists a satisfying assignment. This means that variables from the expression have such values that the expression evaluates to true. The expression is a contradiction if it is not satisfiable. If the expression evaluates to true under all assignments, it is valid.

A naive SAT solver would enumerate all assignments until a satisfiable one is found. However, as this approach has its limitations, other algorithms have emerged, for instance the DPLL algorithm [60]. Currently, modern solvers are based on the CDCL [61] evolution of DPLL.

SAT solvers accept the input formula in the *Conjunctive Normal Form*, so it is usually the case that the encoded problem has to be transformed into *CNF*.

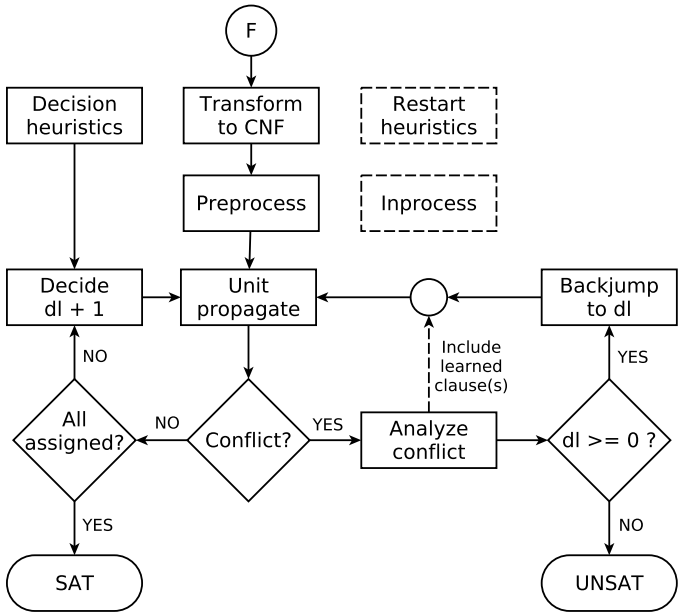


Figure 5.1: CDCL: An overview of the conflict-driven clause learning algorithm.

Every propositional formula can be transformed into an equisatisfiable CNF formula, and there are different methods to do this [62]. After transformation, a preprocessing step can be applied to potentially simplify the input formula [63].

The solver spends most of the time in a procedure called Unit Propagation which is also sometimes called Boolean Constraint Propagation. Unit propagation [64] repeatedly applies the unit clause rule for all unit clauses, either until all implications are exhausted or a conflict occurs. If no conflict is encountered, the solver checks whether all variables are assigned. If so, the algorithm can terminate and return SAT. Otherwise, a decision heuristic is applied to choose a variable on which the search will branch and the decision level *dl* is incremented. However, if a conflict does occur during unit propagation, it is analyzed in order to produce a learned clause. This learned clause is appended to the original formula to aid the search by narrowing the search space.

The conflict analysis algorithm decides the decision level where the CDCL algorithm has to backjump to. If a conflict was detected at the 0-th level, conflict analysis sets $dl = -1$ and the CDCL algorithm returns UNSAT in the next step. Otherwise, the algorithm backjumps to the level dl, or in other words, the assignments made after this level are unset, after which unit propagation is applied again. So, in order to use the power of solvers to perform planning, we have to *encode* the planning problem as a satisfiability problem.

5.3 Encoding

In planning as satisfiability, a planning problem is encoded into a logical formula F_j, where j represents the number of plan steps. This formula is then tested for satisfiability by a solver, which either returns UNSAT for a failed planning attempt or it returns SAT and provides a satisfying assignment which is parsed to yield a plan. If the result from the solver is UNSAT, the planning problem is encoded into a logical formula of length $j + 1$ and tested again by the solver. This goes on sequentially until the solver returns SAT with a satisfying assignment, or until a limit on the plan length is breached.

Example

Let's look at a simple pick and place problem where a robot can move a product between two positions. In order to model this problem, two variables are enough to keep track of the position of the robot and the position of the product.

```
robot_at = {a, b, home}
prod_at = {a, b, grip}
```

Now let's model behavior for this problem with some transitions. We need three types of transitions, namely to move the robot, to pick the product and to place the product. For example, in order to execute a transition that moves the robot to a position *a*, the robot shouldn't be at position *a*. After the move has been executed, the value of variable *robot_at* is updated to that position.

```
for pos in {a, b, home} {
    Transition::new(
        name: "move_to_{pos}",
        guard: robot_at != {pos},
        update: robot_at = {pos}
    )
}

for pos in {a, b} {
    Transition::new(
        name: "pick_at_{pos}",
        guard: robot_at == {pos} AND prod_at == {pos},
        update: prod_at = grip
    )
}

for pos in {a, b} {
    Transition::new(
        name: "place_at_{pos}",
        guard: robot_at == {pos} AND prod_at == grip,
        update: prod_at = {pos}
    )
}
```

For a model containing these seven transitions and for the initial and goal states I and G, calling a planned would yield the following plan:

```
I = robot_at == home AND prod_at == a
G = prod_at == b
Plan = [move_to_a, pick_at_a, move_to_b, place_at_b]
```

In order use solvers to calculate this plan for us, we have to encode this planning problem as a satisfiability problem. To do that, we use time steps to mark the variables, which essentially produces a new set of Boolean variables for each time step. A simple encoding for a plan of length i looks like this:

$$F_i = I_0 \wedge G_i \wedge \delta(T_i)$$

where I_0 are clauses that encode the initial step, G_i are clauses that encode the goal step, and $\delta(T_i)$ are clauses that encode the transitions and the rules

that declare how those transitions can be taken in each step. For the sake of simplicity, lets consider that $\delta(T_i)$ encodes that exactly one transition is to be taken in one step. A single transition t_n for a step i is encoded like this:

$$t_{ni} = t_n.name_i \wedge g_i \wedge e_{i+1}$$

where $t_n.name_i$ keeps track of the name of the transition that is taken in a step, g_i encodes the guard of the transitions for the current step, and e_{i+1} encodes the effect of the taken transition for the next step. To begin, we encode a formula which says that for the given model, there exists a plan of length 0, and feed it to the solver:

$$F_0 = robot\_at\_home_0 \wedge product\_at\_a_0 \wedge product\_at\_b_0$$

We would expect the solver to return UNSAT as there is no sense that a product can be at two different positions at the same time. However, the solver returns an *anomalous* model [56] with the following partial assignment:

```
robot_at_home_0 == true
prod_at_a_0 == true
prod_at_b_0 == true
```

It turns out that we forgot to assert that our initial variable *prod_at* with the domain $\{a, b, grip\}$ can't have more than one value at a time. An option to fix this would be to extend the initial state with an additional conjunct $prod\_at\ != b$. Another fix would be to add *invariants* specifying that a product can't be at two different positions at the same time, or even to make this rule general for all variables. For example, a variable x with a domain $\{a, b, c\}$ can have only one value in a time step. Encoding this rule as a Boolean formula would look something like:

$$(xa_i \wedge \neg xb_i \wedge \neg xc_i) \vee (\neg xa_i \wedge xb_i \wedge \neg xc_i) \vee (\neg xa_i \wedge \neg xb_i \wedge xc_i)$$

After adding this additional constraint, the solver fails to find a satisfiable assignment and returns UNSAT as expected. Our next step is to encode a logical formula that says that for the given model, there exists a plan of length 1, and give that to the solver:

$$F_1 = robot\_at\_home_0 \wedge product\_at\_a_0 \wedge product\_at\_b_1 \wedge \delta(T_1)$$

where $\delta(T_1)$ holds constraints about the transitions and other constraints that disallow anomalous models. If we have encoded the problem correctly, the solver should respond with UNSAT up until we reach time step 3, when a satisfiable assignment is found. After some parsing and filtering we get the expected result:

```
Plan = [move_to_a, pick_at_a, move_to_b, place_at_b]
```

This sequential SAT planning algorithm is shown as Algorithm 1, where certain details like invariant encoding and keeping unchanged values are omitted for brevity.

Algorithm 1: ***Sequential***

Input: *(i, g, M, s_{max})*
Output: *planning_result*

```
1  let step := 0;
2  while step ≤ s_max do
3  |  let ctx := new_context;
4  |  add constraint (ctx, i, 0);
5  |  add constraint (ctx, g, step);
6  |  for s in (0 to step) do
7  |  |  add constraint (ctx, disjunction(M.trans), s);
8  |  |  add constraint (ctx, exactly_one(M.trans.name), s);
9  |  end
10 |  if check(ctx) == UNSAT then
11 |  |  step += 1;
12 |  else
13 |  |  let planning_result = parse(ctx.get_model);
14 |  |  return planning_result;
15 |  end
16 end
17 return new_empty_planning_result;
```

5.4 Speed

Intelligent automation systems have to adapt, react, and deliberate as fast as possible, as an assembly or preparation cycle time is just not long enough to allow planners to spend more than a few seconds to generate new plans. Moreover, the uncertainties from human-robot collaboration or operations such as picking items that have undefined orientations, often means the need to re-plan many times during such operations.

In order to investigate methods that speed up planning, we limit ourselves to classical and deterministic planning problems [23], which we reduce to propositional satisfiability. Considering the nature of problems we try to solve, we are investigating systematic search methods, however, for a comprehensive study on stochastic search algorithms for SAT, refer to [65]. Lastly, instead of look-ahead search methods [66], our interest is shifted towards the CDCL [61] evolution of DPLL [60]. The reason for this is that, again, CDCL performs better on the family of problems we are interested in, namely planing and industrial.

While some solvers such as [67] and [68] do excel in solving hard random instances, they are usually not competitive on structured instances generated from real applications [69]. Still, some research indicates that there might be a way to utilize the strengths of both solver flavours, namely to use look-ahead search as a means to guide a CDCL solver [70].

Big performance improvements in planning as satisfiability can be unlocked when methods are altered on the low, SAT solving level. Some of these methods improve decision heuristics [71], restart heuristics [72], and data structures [73]. We refer to these methods as *low level* methods since they affect the performance of the solver itself, and as such, they are of major interest in the field of planning as satisfiability.

In this study however, we investigate and compare several methods for planning as satisfiability, without getting into the detailed tuning of these solvers. Instead, we focus on what we call *high level* methods to improve the planning performance, which for example include the structure of the model and how to call the solvers. The methods that we investigate are incremental planning [74], adding planning invariants [75], modeling with equality logic [76] and solving with SMT solvers [77], skipping planning steps [78], and subgoaling [23]. The overview of these methods is shown in Figure 5.2.

These methods and their final results are summarized here, however, for a

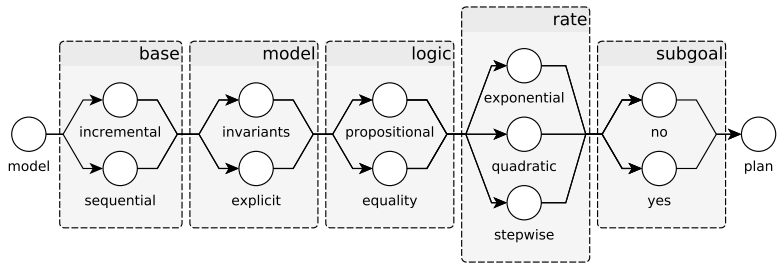


Figure 5.2: There is a total of 48 method combinations, however, not all combinations are investigated in this study.

more comprehensive study, refer to Paper C, where each subsection explores some methods and compares their performances on a set of classical planning benchmarks. The benchmarks that are chosen to test the methods are: *gripper* [79], *blocksworld* [25], *rovers* [80], *barman* [81] and *childsnack* [82]. These benchmarks are chosen to test different strengths and weaknesses of the following planning methods:

1. *Incremental planning:* The downside of Algorithm 1 is that for every iteration where an assignment is not found, a new context has to be created. Gocht and Balyo showed in 2017 [74] that it is possible to achieve a significant speed-up by using an incremental SAT-solver. Instead of throwing away the context with all the accumulated data from previous results, the same context is used and constraints are just added on top. An incremental planning algorithm utilizes an incremental solver which makes it possible to add a *backtracking point* in each step, so that the solver can choose which part of the context to save, and thus learn from previous attempts. In summary, the advantages of an incremental base solver are that learnt clauses are kept, heuristic data is gathered, and that the overhead from asserting the same clauses is reduced.

2. *Adding planning invariants:* Invariants can be considered both as a modeling aid as well as a performance increasing method in automated planning. Invariants are specifications that must hold in every step of the calculated plan. In essence, invariants prune states from the state space. They can be added to the model to forbid some undesired behavior, or to

enforce tighter constraints and thus derive a more compact state space representation. In each case, the state space gets reduced and thus planning is more efficient. For an existing problem, invariants can sometimes be synthesized to speed up planning [75]. Invariants can also often be a convenient tool for modelling different problems, but can in some cases be quite hard to use.

3. *Equality logic and SMT solvers:* Coupling a SAT solver with theory solvers, for example theories such as linear arithmetic, bit vectors, or arrays, SMT based planning techniques [76] can encode and tackle real-world scenarios and complex application domains [83]. By using different *first-order* logic theories to increase expressiveness, some problems can be modeled in a much more convenient way compared to using pure propositional logic. In this study, only equality logic is investigated since it is appropriate for modelling most classical planning problems. As both propositional and equality logic are NP-complete [84], [85], means that they can model the same decision problems with not more than a polynomial difference in variables [86]. Certain problems are more conveniently modeled in equality logic compared to propositional logic, and for some other problems the opposite is true. As for efficiency, the high level structure of the input equality logic formula can potentially be used to make the decision procedure work faster. This information may be lost if the problem is modelled directly in propositional logic [86].

4. *Skipping steps:* The planning algorithms increment the plan length by one when an assignment is not found, after which the encoding for that length is tested for satisfiability. This is a good method when the yielded plan should be of minimal length. However, calculating the shortest length plan can sometimes be very slow, as Rintanen showed in [87]. The evaluation cost of an unsatisfiable formula can be much higher than the evaluation cost of a satisfiable one, even if the latter is not of shortest length. Especially, when considering the sum of evaluation costs for all unsatisfiable formulas, the planner can spend a lot of time trying to find a satisfiable assignment. This is because the cost of evaluating the unsatisfiable formulas usually increases exponentially as the plan length increases [87]. When finding the first satisfiable solution which yields the plan with the minimal length is not a strict requirement, an

improvement in the planning time can sometimes be achieved. Rather than increasing the plan length by *one* after an assignment is not found, this improvement can be realized by incrementing the plan length by a larger value. This allows us to skip some hard unsatisfiable instances which take a long time to evaluate.

5. *Subgoaling:* When a goal is defined as a conjunction of several predicates, such predicates can be looked at as *subgoals*. Instead of asking the planner to reach the monolithic goal in one planning task, multiple planning tasks can be instantiated to plan for each subgoal. Having a simpler goal shortens the plan length and thus the planning time. As mentioned before, the cost of evaluating unsatisfiable formulas increases exponentially, thus it is usually faster to search for a large number of shorter plans than the other way around. If the subgoal order is not correct, finding a plan can sometimes be slower or even impossible. This depends quite much on the nature of the problem itself as planning problems often exhibit symmetry properties that could be exploited to speed up their solving.

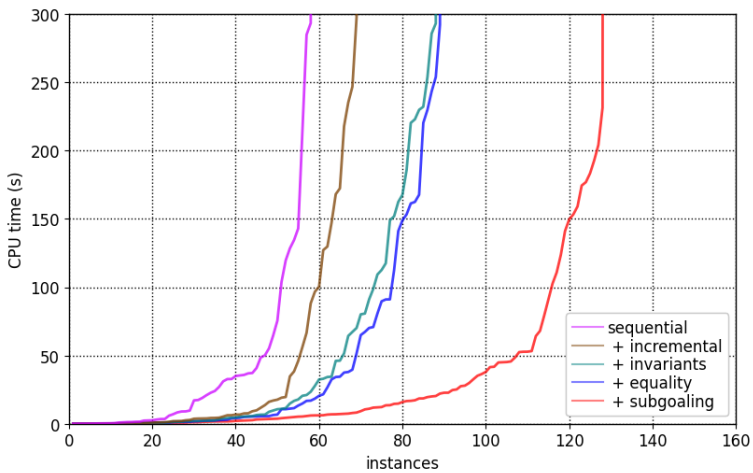


Figure 5.3: High-level planning methods contribution to performance.

The benchmarks were ran on an Optiplex 9020 desktop PC with 8GB of RAM and an Intel Core i7-4790 CPU clocked at 3.60GHz. All algorithms used in this study were implemented using Z3's [77] quantifier free finite domain (QF_FD) theory, which supports propositional logic, bit-vector theories, pseudo-Boolean constraints, and enumeration data types. Figure 5.3 shows how the studied methods contributed to increased planning performance.

5.5 A compositional approach

A known issue with SAT-based automated planning is that plan calculation seems to slow down significantly as the plan length increases [57]. In an effort to avoid this limitation while utilizing the strengths of SAT-based planning, a compositional algorithm is presented here that divides a planning problem into a number of simpler problems that are faster to solve. This is done with a combination of abstraction refinement using activation parameters and step-wise problem generation, resolution and concatenation. The proposed algorithm is tested on a few examples, where it is shown that a significant speed-up [87] can sometimes be achieved, especially for highly symmetrical problems.

The compositional algorithm with its final results and additional insights is summarized here, however, for a more comprehensive study, refer to Paper D. To start with, a simple incremental planning algorithm based on [74] is utilized in to solve individual problems generated by the compositional algorithm. In order for the compositional algorithm to refine, generate and solve parts of the complete planning problem, we have to allow its abstraction and refinement.

The planning problem is *parameterized* so that the compositional algorithm can enable or disable certain predicates in order to generate abstracted input constraints to the incremental algorithm. In order to do this, every variable defined in the problem is assigned a parameter, based on which they are grouped. These parameters allow the compositional algorithm to turn these groups, i.e. parts of the model, *on* or *off* before sending the problem to the incremental algorithm to be solved.

Initially, all parameters are turned off, and they are activated one by one, or by some other rule based on a refinement order. After each activation, the compositional algorithm generates a number of problems based on the trace of the previous level, sends them to the incremental algorithm to be solved and

concatenates the results in order to get the next level. After all parameters are activated, the algorithm terminates and yields a plan.

Results

It is shown on an example that utilizing this algorithm can not only reduce planning time compared to other non-decomposing approaches, but also plan length compared to other sub-optimal algorithms such as subgoaling. A highly symmetrical problem exemplifies this result.

This problem, from the International Planning Competition [88], is about a mobile robot with two grippers which can carry a ball in each. There are two adjacent rooms, and the goal is to take N balls from one room to another. There are many problem instances for this example, ranging from 1 to 40 balls. A timeout of 160 seconds is set to interrupt the planning. In this example, the performance of algorithms (marked as *other*) from in Paper C are compared to the subgoaling and compositional algorithm with two differently defined variable grouping, as shown on Figure 5.4.

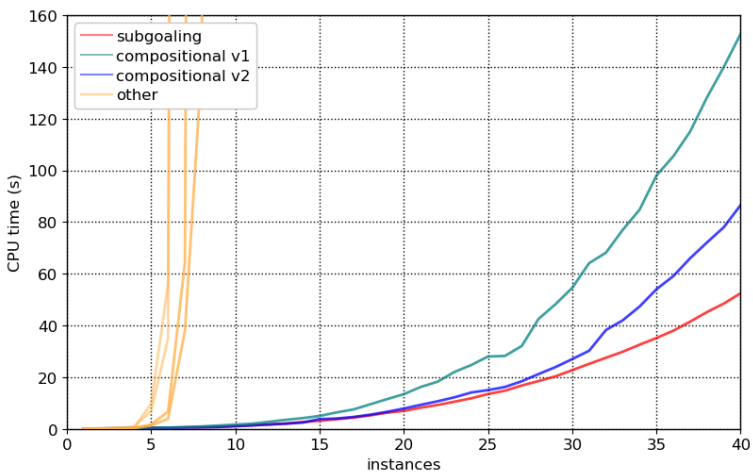


Figure 5.4: Compositional vs. other algorithms on the gripper benchmark.

The incremental algorithm, behaving like breadth-first search, finds the optimal plan, meaning that the robot takes a ball in each gripper before moving to the other room. However, because of the combinatorial explosion

occurring with each added ball, the incremental algorithm quickly exceeds the timeout.

The subgoaling algorithm looks at each conjunct in the goal predicate as a separate goal. As the problem is highly symmetrical, the execution order of such goals does not matter. While solving them, the algorithm behaves like depth-first search, meaning that the robot will pick just one ball before moving to the other room to place it. The subgoaling algorithm can find plans for this problem very fast, however, being that the robot moves only one ball at the time, the solution quality is not so good.

Let's see how the compositional algorithm fares. In the first compositional approach, marked as *v1* in Figure 5.4, the variables are grouped so that the robot variables go to one group, the gripper variables to another, and each ball variable goes to a separate group. This produces the same plan quality as the subgoaling algorithm as it does virtually the same thing.

The second compositional approach puts two balls in a group instead of one, and these results are marked with *v2* in Figure 5.4. Now, there is a better compromise between the amount of planning tasks and the length of each result, and as shown on the right side of the following listing, the plan quality is improved:

```
subgoaling and comp. v1:          incremental and comp. v2:
r1_pick_b1_in_room_a_with_gl      r1_pick_b1_in_room_a_with_gl
move_r1_to_b                      r1_pick_b2_in_room_a_with_gr
r1_drop_b1_in_room_b_from_gl      move_r1_to_b
move_r1_to_a                      r1_drop_b1_in_room_b_from_gl
r1_pick_b2_in_room_a_with_gl      r1_drop_b2_in_room_b_from_gr
move_r1_to_b                      move_r1_to_a
r1_drop_b2_in_room_b_from_gl      r1_pick_b3_in_room_a_with_gr
move_r1_to_a                      r1_pick_b4_in_room_a_with_gl
r1_pick_b3_in_room_a_with_gl      move_r1_to_b
move_r1_to_b                      r1_drop_b3_in_room_b_from_gr
r1_drop_b3_in_room_b_from_gl      r1_drop_b4_in_room_b_from_gl
move_r1_to_a                      move_r1_to_a
r1_pick_b4_in_room_a_with_gl      r1_pick_b5_in_room_a_with_gr
move_r1_to_b                      move_r1_to_b
r1_drop_b4_in_room_b_from_gl      r1_drop_b5_in_room_b_from_gr
move_r1_to_a
```

```
r1_pick_b5_in_room_a_with_gr
move_r1_to_b
r1_drop_b5_in_room_b_from_gr
```

Drawbacks

Even if grouping the variables was done correctly, the order in which the variables are activated can substantially influence the plan quality and planning speed. Solving the same problem with a different refinement order can give quite different plans, as shown on the left side of Figure 5.5.

If there are several valid plans of different lengths that can be calculated at a certain step, a shortest one will be yielded. However, this might produce a longer plan down the line of refinement, as it is shown on the right side of Figure 5.5.

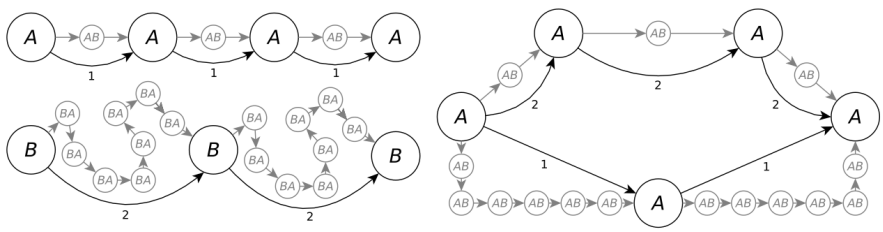


Figure 5.5: Potential drawbacks of the compositional algorithm.

CHAPTER 6

Concluding remarks and future work

In this thesis, we have discussed the need for intelligent automation systems, at least from a perspective gained while developing such solutions at Volvo Trucks. One of such solutions was used throughout this thesis to exemplify a framework for virtual preparation and commissioning, as well as to investigate the various implementation possibilities of resource, communication and control architectures. Lastly, as intelligent automation systems depend on automated planning, an investigation was made on the various planning methods in order to determine how each of them contribute to an increase in planning performance. This chapter concludes the thesis by trying to answer the research questions introduced in Section 1, as well as start a discussion about possible future work.

RQ1 Why do we need intelligent automation systems?

Modern production faces many challenges, many of which will need a lot of time to overcome. Some of these challenges were identified in Chapter 1, so let's try to summarize ways to address them:

1 *Variability*: Intelligent production systems should be flexible and able to accommodate product variability. Automated planning and verification algorithms at the core of intelligent systems should handle substantial product variability by removing the need to explicitly program control code.

2 *Speed*: Intelligent production systems should adapt and easily integrate new products and production scenarios. Moreover, the core algorithms should make it possible to quickly change scenarios, adapt models, and exercise safety, all without the need to halt production.

3 *Cost*: Contrary to traditional production systems, intelligent systems should cost a fraction of their price to implement and maintain. This is due to the fact that operator safety should now be assured while being close to collaborative robots, i.e. there are no extra safety measures that have to be implemented. Moreover, operators should take the role of workstation programmers and maintainers, further lowering the cost.

4 *Society*: Intelligent production systems should take care of their workers. As robots and humans now work together, robots should perform the repetitive and injury prone tasks while humans do the very delicate and tactile tasks. Moreover, as discussed, the right way to achieve an optimal flow while minimizing cost should be to have both robots and operators working together.

5 *Sustainability*: Online planning and optimization algorithms of intelligent automation solutions and the flexible topology of production systems should make it possible to implement just-in-time production. There should be no more need to manufacture and store large quantities of products and material.

6 *Legacy*: Intelligent systems should be flexible not only on their own, but also in conjunction with legacy manufacturing. It should be possible to easily integrate intelligent subsystems into an existing legacy system.

RQ2 *What to consider when implementing architectures for intelligent automation systems?*

There is no one solution that can best fit all scenarios, thus engineers have to think about the benefits and drawbacks of implementing an architecture for a specific scenario. The main things to consider are the need for flexibility and human-robot collaboration, fault frequency and tolerance, and restart ability. This will determine if state has to be decentralized, or in other words, if certain resources can, or have to, be implemented as stateless, stateful, or intelligent. Moreover, one has to consider if the communication with those resources should be state or event based, and finally, if the control architecture should be designed as centralized, distributed, or hierarchical.

RQ3 *How to support the development of intelligent automation systems with tools such as virtual preparation, commissioning, and testing?*

Developing and controlling highly flexible and collaborative systems is hard because it is quite a novel approach to production. The core of this problem is the missing infrastructure with all the tools, frameworks, examples, perspectives, and lessons learned. Thankfully, we are not entirely rolling in the dirt, as the ROS community has worked hard for years to provide us a set of very useful tools and a good sense of what is possible out there. That being said, such tools and solutions are not always usable out of the box, and quite often, frameworks, drivers, simulations, etc., have to be developed either for a general, or a specific use-case.

RQ4 *What makes intelligent automation systems intelligent and how can we make intelligent reasoning fast enough?*

The core of intelligent automation systems is a SAT solver, which is used to calculate satisfiable assignments for state variables. For planning and deliberatively acting towards goals, a SAT solver is employed to solve a planning problem encoded into a Boolean formula. As it is argued in Chapter 5 of this thesis, planning has to be as fast as possible in order to be usable in real industrial scenarios, where planning tasks are instantiated very often.

Some of the evaluated high-level methods increase performance more than other methods do, and some can even ease the modeling process.

However, the biggest observed increase was gained when trying to decompose a planning problem into a multitude of problems using either the subgoaling or the compositional algorithm, especially for highly symmetrical problems. As such, the solver does not spend a lot of time trying to solve large unsatisfiable instances generated from the monolithic model.

Future work

When implementing an intelligent automation system, one of the most challenging tasks is developing a correct model. A lot of possible behavior and constraints can be missed or falsely programmed while modeling, leading to empty or anomalous planning results. It is very hard to anticipate the consequences of a faulty model, as it is often the case that the possible planning traces are, simply put, untraceable for a human brain.

As the complexity of the system grows, so does the complexity of the model, which demands even more cognitive effort from the developers. In models of large and complex systems, it is becoming very hard to find and correct mistakes and missing links, and as such, it forces a *trial and error* kind of endgame development. This can sometimes be frustrating, especially if a minor change in the model breaks the complete behavior just a few hours before a demonstration has to be made.

An interesting step in future work could be to investigate how to better aid engineers when developing a model. As presented in this thesis, the integrated virtual preparation and commissioning framework provides tools to help develop and test models. However, in cases where plans can not be found, the developers are given no feedback about what is wrong, and they are forced to scroll through the model in an effort to find errors. Needless to say, this is quite strenuous and time consuming.

Since we are using SAT solvers for automated planning, identifying errors in the model from extracted minimal unsatisfiable cores could probably give us some insight. Moreover, using property based testing in conjunction with planning based on satisfiability during the development stage could ideally provide us with hints on how to fill in the gaps of a boilerplate model, further easing the modeling process.

CHAPTER 7

Appendix

Automation 2.0

Production is not the only activity that can be automated. In order to paint a full picture of changes in automation, we have to talk about other automatable activities and the effects automation has on other aspects of life. More importantly, the potential effects of changes in automation for the generations to come. In fact, as automation researchers, it is our duty to do so.

Transportation, services, medicine, and even entertainment is being automated as we speak using the latest iteration in human innovation, machine learning algorithms. Learning algorithms do exactly the thing which their name suggests, they learn. Either supervised or unsupervised, such algorithms feast on huge amounts of data that is constantly being collected from other machines, algorithms, and humans. Using this big data, learning algorithms are able to infer control rules for almost every application imaginable. From this comes the big question: Could this new kind of automation make companies think twice before spending a lot of money on a bunch of skilled engineers, journalists, analysts, lawyers, teachers, doctors and even artists?

The scientific community is split between two views about the shape of

things to come. Some believe that the trend of changing for better jobs will continue as it always has until now, while others support a view that the number of newly generated jobs will not be able to compensate for the number of lost jobs and the population growth.

A study from 2013 assessed the probability of job automation for 702 different occupations in the US [89], which is 97% of its total workforce. The conclusion is that a staggering 47% of total US employment is at high risk of being automated relatively soon. Moreover, the authors draw quite compelling connections in the literature which makes this work well worth reading.

On the other hand, there are studies which suggest that such assessments are overestimating the number of automatable jobs. For example, a study from 2017 estimates the number of automatable jobs in the US to be around 9% [90]. The authors explain this unexpectedly low number to originate in the heterogeneity of tasks within occupations. Namely, they take into account the vast variation of tasks within one occupation, which the study from 2013 does not.

Another study, which is based on [89] and the Survey of Adult Skills (PIAAC) for the 32 OECD countries that have participated in it, estimates the number of jobs which are at high risk of automation to be at around 14% [91]. Similarly to [90], this study uses individual level data which could explain this much lower estimate of automatable jobs.

However, in an interesting follow-up discussion [92], it is pointed out that the authors of [90] and [91] did not base their research only on tasks, but also other data such as firm characteristics, personal data and individual worker differences. According to them, the likelihood of a job being automated also depends on factors such as sex, education, age, and income. As you can imagine, not all can agree that this is the case.

Economics 2.0

Historically, it has been common to blame progress in automation for the rising unemployment rate. Even more recently, some researchers have suggested that automation could explain the 2007 - 2010 rapid increase in unemployment [93]. However, if you look at the current numbers and try to look past the calamity which Covid-19 has caused, the unemployment rate was never lower [94].

Still, some theorists are hard at work in trying to warn us about the effects

of progress in automation can have for our society in the future [95]. According to them, it looks like we are approaching a paradoxical point where the algorithms generate an abundance of cheap resources and services, yet unaffordable for the majority of people that will be left unemployed by the same algorithms. A zero marginal cost society.

Some argue that the spoils of automation will be shared more unequally than ever [96]. Even now, it is visible that some of the wealthiest companies in the world employ a disproportionately small amount of people. The current Covid-19 crisis did not help at all, in fact, strong companies have become even stronger which made the gap between rich and poor even bigger.

An idea suggests radical taxing strategies, for instance such as imposing an additional *robot tax* based on the amount of automated work within a company [97]. This would either force companies to expand their workforce as the company is growing in order to achieve an optimal increase in the company's gross income, or payout enormous tax money which could be used to re-skill or re-train displaced workforce. There are also those who warn that taxing automation could disrupt the economy, as such a hard policy could put an overall halt on progress [98].

Another idea suggests enforcing a universal basic income, which is a regular payment given to everyone in society to create a minimum income floor [99]. People would still have the opportunity to work, but in contrast to welfare, a universal basic income would not be cancelled if the person would start working, rather, the salary would be added on top.

Ethics 2.0

It is not a secret that self-driving cars are slowly but surely making their way into everyday use. It is expected that the global autonomous car market is going to reach $1.38 trillion in 2025 [100]. We work hard to research and develop technology that will make self-driving cars reality, yet we rarely ask ourselves what is the reason to do that. Is it to lower air pollution, reduce traffic congestion and offer people a way to scroll through their Instagram feed while their car is driving them to work? Or is it just because *we can*?

What we can hope for at least, is that autonomous driving will make our roads safer. In 2018, the World Health Organization has reported that the number of annual road traffic deaths has reached 1.35 million. Let that number

sink in together with the fact that road traffic injuries are now the leading killer of people aged 5-29 years [101]. We are bad drivers, and self-driving cars don't need to be perfect in order to drastically reduce that number, they just have to be better than us. In fact, they already are.

Is this enough to push for a global replacement of human drivers? This could have been an extreme example, but it puts into perspective some of the ethical questions raised every day. For instance, in production, is it OK to automate a part of manual assembly which is quite repetitive and injury prone? [102] Or maybe, how can we develop and deploy reliable intelligent applications that can guarantee consistently accurate and unbiased behavior? [103]

CHAPTER 8

Summary of included papers

This chapter provides a summary of the included papers.

8.1 Paper A

Endre Erős, Martin Dahl, Atieh Hanna, Per-Lage Götvall, Petter Falkman and Kristofer Bengtsson
Development of an Industry 4.0 demonstrator using Sequence Planner and ROS2
Published in Robot operating system (ROS): The complete reference vol. 5, pp. 3–29, 2021.
Studies in Computational Intelligence book series (SCI, volume 895)

DOI: https://doi.org/10.1007/978-3-030-45956-7 .

This paper presents the development of an industrial demonstrator and the control infrastructure Sequence Planner, together with some practical development guidelines and lessons learned. The demonstrator includes robots, smart tools, human-machine interfaces, online path and task planners, etc.

8.2 Paper B

Endre Erős, Martin Dahl, Atieh Hanna, Anton Albo, Petter Falkman and Kristofer Bengtsson
Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system
Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)
vol. 24, pp. 407–413, 2019.

DOI: 10.1109/ETFA.2019.8869444 .

This paper discusses integrated virtual preparation and commissioning (IVPC) of an engine assembly station. The purpose of virtual commissioning is to enable the control software, which controls and coordinates different devices in a production process, to be tested and validated before physical commissioning. To support IVPC in intelligent automation systems, ROS2 components are auto-generated to continuously test and improve the model in Sequence Planner.

8.3 Paper C

Endre Erős, Martin Dahl, Petter Falkman and Kristofer Bengtsson
Evaluation of high level methods for efficient planning as satisfiability
Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)
vol. 26, 2021.

DOI: 10.1109/ETFA45728.2021.9613254 .

This paper investigates and compares several methods for planning as satisfiability. It does not however study the detailed tuning of SAT solvers, but instead focuses on high level methods to improve planning performance, which for example includes the structure of the model and how to call the solvers.

8.4 Paper D

Endre Erős, Martin Dahl, Petter Falkman and Kristofer Bengtsson
Towards compositional automated planning
Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) vol. 25, pp. 416–423, 2020.

DOI: 10.1109/ETFA46521.2020.9212040 .

This paper presents an implementation of a high-level compositional algorithm that utilizes an incremental SAT solver as its planning engine. This is achieved by generating abstractions from the main parameterized planning problem, solving them using the incremental solver, and concatenating the results in order to get a complete plan.

References

[1] D. Bennett, "Future challenges for manufacturing," English, *Journal of Manufacturing Technology Management*, vol. 25, no. 1, pp. 2–6, Dec. 2014, ISSN: 1741-038X.

[2] A. Alptekinoglu and C. J. Corbett, "Mass customization vs. mass production: Variety and price competition," *Manuf. Serv. Oper. Manag.*, vol. 10, pp. 204–217, 2008.

[3] H. C. Dreyer, R. Bjartnes, T. H. Netland, and J. O. Strandhagen, "Real-time supply chain planning and control - a case study from the norwegian food industry," 2008.

[4] K. Butner, "The digital overhaul: Rethinking manufacturing in the digital age," in *IBM Institute for Business Value*, 2015.

[5] G. I. Fragapane, D. A. Ivanov, M. Peron, F. Sgarbossa, and J. O. Strandhagen, "Increasing flexibility and productivity in industry 4.0 production networks with autonomous mobile robots and smart intralogistics," *Annals of Operations Research*, pp. 1–19, 2020.

[6] M. Dahl, *Preparation and control of intelligent automation systems.* Gothenburg: Chalmers tekniska högskola, 2021, ISBN: 9789179054465.

[7] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[8] L. S. Gottfredson, "Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography," *Intelligence*, vol. 24, no. 1, pp. 13–23, 1997, Special Issue Intelligence and Social Policy, ISSN: 0160-2896.

[9] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*, 1st. O'Reilly Media, Inc., 2015, ISBN: 1449323898.

[10] P. Bergagård, *On restart of automated manufacturing systems*. Gothenburg: Chalmers tekniska högskola, 2015, ISBN: 9789175971261.

[11] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica–an efficient tool for large-scale discrete event systems," *IFAC-Papers On Line*, vol. 50, no. 1, pp. 5794–5799, 2017, 20th IFAC World Congress, ISSN: 2405-8963.

[12] *Virtual Commissioning of Vehicle Maintenance Operations: Unification*, `https://www.vinnova.se/en/p/virtual-commissioning-of-vehicle-maintenance-operations-unification/`, [Online; accessed 06-Dec-2021].

[13] *Vision and Practice at Volvo Group GTO: Industry 4.0 and PLM in Global Truck Manufacturing*, `https://www.engineering.com/story/vision-and-practice-at-volvo-group-gto-industry-40-and-plm-in-global-truck-manufacturing`, [Online; accessed 07-Dec-2021].

[14] *Sustainable, Peaceful and Efficient Robotic Refuse Handling: Unicorn*, `https://www.vinnova.se/en/p/unicorn---sustainable-peaceful-and-efficient-robotic-refuse-handling/`, [Online; accessed 07-Dec-2021].

[15] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, pp. 485–493, Jun. 2018.

[16] *Man to machine interface in material handling*, `https://www.mhlnews.com/labor-management/article/22047551/manmachine-interface-in-material-handling`, [Online; accessed 05-Sep-2021], 2021.

[17] T. Foote, "Tf: The transform library," in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1–6.

[18] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics and Automation Magazine - IEEE ROBOT AUTOMAT*, vol. 19, pp. 18–19, Mar. 2012.

[19] M. Cashmore, M. Fox, D. Long, *et al.*, "Rosplan: Planning in the robot operating system," in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'15, Jerusalem, Israel: AAAI Press, 2015, pp. 333–341, ISBN: 978-1-57735-731-5.

[20] F. Rovida, M. Crosby, D. Holz, *et al.*, "Skiros—a skill-based robot control platform on top of ros," in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 121–160, ISBN: 978-3-319-54927-9.

[21] E. Aertbeliën and J. De Schutter, "Etasl/etc: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 1540–1546.

[22] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 564–571.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*, 1st. USA: Cambridge University Press, 2016, ISBN: 1107037271.

[24] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *ArXiv*, vol. abs/1106.4561, 2003.

[25] F. Bacchus, "Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems," *AI Magazine*, vol. 22, no. 3, p. 47, Sep. 2001.

[26] *opcua ros2 bridge*, `https://github.com/sequenceplanner/opcua_ros2_bridge`, [Online; accessed 20-Dec-2021], 2021.

[27] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, vol. 1, 2001, 1/121–1/132 vol.1.

[28] M. Dahl, E. Erős, K. Bengtsson, M. Fabian, and P. Falkman, "Sequence planner: A framework for control of intelligent automation systems," *Submitted to Robotics and Conputer-Integrated Manufacturing*, 2021.

[29] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28 573–28 593, 2018.

[30] P. Leitão, N. Rodrigues, C. Turrin, and A. Pagani, "Multiagent system integrating process and quality control in a factory producing laundry washing machines," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 879–886, 2015.

[31] M. Dubois, Y. Guyadec, and D. Duhaut, "Masl: A language for multiagent system," *Multiagent Systems*, Jan. 2009.

[32] Y. barzegari, J. Zarei, R. Razavi-Far, and M. Saif, "Consensus of first order multi-agent systems with actuator or dynamic fault by weighted adjacency matrix approach (wama)," in *2021 7th International Conference on Control, Instrumentation and Automation (ICCIA)*, 2021, pp. 1–6.

[33] L. Monostori, J. Váncza, and S. R. Kumara, "Agent-based systems for manufacturing," *CIRP annals*, vol. 55, no. 2, pp. 697–720, 2006.

[34] L. Wang, R. Gao, J. Váncza, *et al.*, "Symbiotic human-robot collaborative assembly," *CIRP Annals*, vol. 68, no. 2, pp. 701–726, 2019, ISSN: 0007-8506.

[35] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018, ISSN: 0957-4158.

[36] A. Hanna, K. Bengtsson, P.-L. Götvall, and M. Ekström, "Towards safe human robot collaboration - risk assessment of intelligent automation," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 424–431.

[37] J. Schuppen, O. Boutin, P. Kempker, *et al.*, "Control of distributed systems: Tutorial and overview," *EUROPEAN JOURNAL OF CONTROL*, vol. 17, pp. 579–602, Sep. 2011.

[38] *Centralised and Distributed Control Systems*, `https://www.theautomationengineer.com/insight/centralised-and-distributed-control-systems/`, [Online; accessed 21-Dec-2021].

[39] A. Daneels and W. Salter, "What is scada?," 1999.

[40] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *The Knowledge Engineering Review*, vol. 19, pp. 281–316, Dec. 2004.

[41] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[42] F. Auinger, M. Vorderwinkler, and G. Buchtela, "Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop"," in *WSC'99. 1999 Winter Simulation Conference Proceedings. 'Simulation - A Bridge to the Future' (Cat. No.99CH37038)*, vol. 1, 1999, 798–805 vol.1.

[43] N. Shahim and C. Møller, "Economic justification of virtual commissioning in automation industry," in *2016 Winter Simulation Conference (WSC)*, 2016, pp. 2430–2441.

[44] S. T. Mortensen and O. Madsen, "A virtual commissioning learning platform," *Procedia Manufacturing*, vol. 23, pp. 93–98, 2018, "Advanced Engineering Education and Training for Manufacturing Innovation"8th CIRP Sponsored Conference on Learning Factories (CLF 2018), ISSN: 2351-9789.

[45] M. Oppelt and L. Urbas, "Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering," *Proceedings, IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2564–2570, Feb. 2015.

[46] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: Supporting formal methods during automation systems development," *IFAC-Papers On Line*, vol. 49, no. 12, pp. 1939–1944, 2016, 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016, ISSN: 2405-8963.

[47] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-Papers On Line*, vol. 51, no. 11, pp. 1016–1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, ISSN: 2405-8963.

[48] *Aruco Tag Detection*, `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`, [Online; accessed 11-Dec-2021].

[49] D. R. MacIver, *Hypothesis 4.24*, `https://github.com/HypothesisWorks/hypothesis`, 2018.

[50] K. Claessen and J. Hughes, "Quickcheck: A lightweight tool for random testing of haskell programs," in *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '00, New York, NY, USA: ACM, 2000, pp. 268–279, ISBN: 1-58113-202-6.

[51] M. Dahl, K. Bengtsson, and P. Falkman, "Application of the sequence planner control framework to an intelligent automation system with a focus on error handling," *Machines*, vol. 9, no. 3, 2021, ISSN: 2075-1702.

[52] M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, "Guard extraction for modeling and control of a collaborative assembly station," *IFAC-Papers On Line*, vol. 53, no. 4, pp. 223–228, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11-13 November 2020, ISSN: 2405-8963.

[53] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518, ISBN: 978-3-540-24605-3.

[54] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X.

[55] S. Akers, "Binary decision diagrams," *Computers, IEEE Transactions on*, vol. C-27, pp. 509–516, Jul. 1978.

[56] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the 10th European Conference on Artificial Intelligence*, ser. ECAI '92, Vienna, Austria: John Wiley & Sons, Inc., 1992, pp. 359–363, ISBN: 0471936081.

[57] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012, ISSN: 0004-3702.

[58] ——, "Madagascar: Scalable planning with sat," *Proceedings of the 8th International Planning Competition (IPC-2014)*, vol. 21, 2014.

[59] ——, "Search methods for classical and temporal planning," *Tutorials of the 21th European Conference on Artificial Intelligence (ECAI 2014)*, vol. 21, 2014.

[60] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962, ISSN: 0001-0782.

[61] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[62] S. Alouneh, S. Abed, M. H. A. Shayeji, and R. Mesleh, "A comprehensive study and analysis on sat-solvers: Advances, usages and achievements," *Artificial Intelligence Review*, pp. 1–27, 2018.

[63] R. Arora and M. Hsiao, "Cnf formula simplification using implication reasoning," Dec. 2004, pp. 129–134, ISBN: 0-7803-8714-7.

[64] K. R. Apt, "Some remarks on boolean constraint propagation," in *New Trends in Constraints*, K. R. Apt, E. Monfroy, A. C. Kakas, and F. Rossi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 91–107, ISBN: 978-3-540-44654-5.

[65] H. Hoos and T. Stützle, "Local search algorithms for sat: An empirical evaluation," *Journal of Automated Reasoning*, vol. 24, pp. 421–481, Jan. 2000.

[66] M. Heule and H. van Maaren, "Look-ahead based sat solvers.," *Handbook of satisfiability*, vol. 185, pp. 155–184, 2009.

[67] A. Anbulagan and C.-M. Li, "Heuristics based on unit propagation for satisfiability problems," Jul. 2000.

[68] O. Dubois and G. Dequen, "A backbone-search heuristic for efficient solving of hard 3-sat formulae," Jan. 2001, pp. 248–253.

[69] L. Zhang and S. Malik, "The quest for efficient boolean satisfiability solvers," in *Proceedings of the 14th International Conference on Computer Aided Verification*, ser. CAV '02, Berlin, Heidelberg: Springer-Verlag, 2002, pp. 17–36, ISBN: 3540439978.

[70] M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and conquer: Guiding cdcl sat solvers by lookaheads," in *Hardware and Software: Verification and Testing*, K. Eder, J. Lourenço, and O. Shehory, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 50–65, ISBN: 978-3-642-34188-5.

[71] J. Rintanen, "Heuristics for planning with sat," vol. 6308, Sep. 2010, pp. 414–428, ISBN: 978-3-642-15395-2.

[72] J. Huang, "The effect of restarts on the efficiency of clause learning," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07, Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2318–2323.

[73] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001 , pp. 530–535.

[74] S. Gocht and T. Balyo, "Accelerating sat based planning with incremental sat solving," in *ICAPS*, 2017.

[75] J. Rintanen, "An iterative algorithm for synthesizing invariants," in *AAAI/IAAI*, 2000.

[76] J. E. Arxer, "Smt techniques for planning problems," 2018.

[77] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3.

[78] J. Rintanen, "Evaluation strategies for planning as satisfiability," *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 682–687, 2004.

[79] D. M. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, no. 2, p. 35, Jun. 2000.

[80] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis.," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 1–59, Dec. 2003.

[81] A. Coles, A. Coles, A. Olaya, *et al.*, "A survey of the seventh international planning competition," *Ai Magazine*, vol. 33, pp. 83–88, Mar. 2012.

[82] M. Vallati, L. Chrpa, M. Grześ, *et al.*, "The 2014 international planning competition: Progress and trends," *AI Magazine*, vol. 36, no. 3, pp. 90–98, Sep. 2015.

[83] A. Bit-Monnot, F. Leofante, L. Pulina, and A. Tacchella, "Smt-based planning for robots in smart factories," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali, Eds., Cham: Springer International Publishing, 2019, pp. 674–686, ISBN: 978-3-030-22999-3.

[84] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71, Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158, ISBN: 9781450374644.

[85] D. Kozen, "Positive first-order logic is np-complete," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 327–332, 1981.

[86] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*, 2nd ed. Springer Publishing Company, Incorporated, 2016, ISBN: 978-3-662-50497-0.

[87] J. Rintanen, "Evaluation strategies for planning as satisfiability," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser.
ECAI'04, Valencia, Spain: IOS Press, 2004, 682–686
, ISBN: 9781586034528.

[88] D. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, pp. 35–55, 2000.

[89] C. B. Frey and M. A. Osborne, "The future of employment: How susceptible are jobs to computerisation?" *Technological Forecasting and Social Change*, vol. 114, pp. 254–280, 2017, ISSN: 0040-1625.

[90] M. Arntz, T. Gregory, and U. Zierahn, "Revisiting the risk of automation," *Economics Letters*, vol. 159, no. C, pp. 157–160, 2017.

[91] L. Nedelkoska and G. Quintini, "Automation, skills use and training," no. 202, 2018.

[92] *Automation and the future of work: understanding the numbers*, `https://www.oxfordmartin.ox.ac.uk/blog/automation-and-the-future-of-work-understanding-the-numbers/`, [Online; accessed 02-Sep-2021], 2018.

[93] E. Brynjolfsson and A. McAfee, *Race Against the Machine: How the Digital Revolution is Accelerating Innovation, Driving Productivity, and Irreversibly Transforming Employment and the Economy*. Digital Frontier Press, 2011, ISBN: 9780984725113.

[94] *OECD: Unemployment rate*, `https://data.oecd.org/unemp/unemployment-rate.htm#indicator-chart`, [Online; accessed 05-Sep-2021], 2021.

[95] J. Rifkin, *The zero marginal cost society : the internet of things, the collaborative commons, and the eclipse of capitalism / Jeremy Rifkin*, English, First edition. Palgrave Macmillan New York, NY, 2014, 356 pages.

[96] E. Brynjolfsson and A. McAfee, *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*, 1st. W. W. Norton; Company, 2014, ISBN: 0393239357.

[97] *Robot tax: The pros and cons of taxing robotic technology in the workplace*, `https://www.futureofworkhub.info/comment/2019/12/4/robot-tax-the-pros-and-cons-of-taxing-robotic-technology-in-the-workplace`, [Online; accessed 05-Sep-2021], 2021.

[98] *The case against robot tax*, `https://itif.org/publications/2019/04/08/case-against-taxing-robots`, [Online; accessed 05-Sep-2021], 2021.

[99] G. Allegri and R. Foschi, "Universal basic income as a promoter of real freedom in a digital future," *World Futures*, vol. 77, no. 1, pp. 1–22, 2021.

[100] *Autonomous Cars Global Market Report 2021: COVID-19 Growth and Change to 2030*, `https://www.researchandmarkets.com/reports/5321395/autonomous-cars-global-market-report-2021-covid`, [Online; accessed 05-Sep-2021], 2021.

[101] *Global status report on road safety 2018*, `https://www.who.int/publications/i/item/9789241565684`, [Online; accessed 05-Sep-2021], 2021.

[102] B. O'Neil, M. E. Forsythe, and W. D. Stanish, "Chronic occupational repetitive strain injury.," *Canadian family physician Medecin de famille canadien*, vol. 47, pp. 311–6, 2001.

[103] *Gartner Says Nearly Half of CIOs Are Planning to Deploy Artificial Intelligence*, `https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence`, [Online; accessed 05-Sep-2021], 2021.

Part II

Papers

PAPER A

Development of an Industry 4.0 demonstrator using Sequence Planner and ROS2

Endre Erős, Martin Dahl, Atieh Hanna, Per-Lage Götvall, Petter Falkman and Kristofer Bengtsson

Published in Robot operating system (ROS): The complete reference vol. 5, pp. 3–29, 2021.
Studies in Computational Intelligence book series (SCI, volume 895)

DOI: https://doi.org/10.1007/978-3-030-45956-7

The layout has been revised.

Abstract

In many modern automation solutions, manual off-line programming is being replaced by online algorithms that dynamically perform tasks based on the state of the environment. Complexities of such systems are pushed even further with collaboration among robots and humans, where intelligent machines and learning algorithms are replacing more traditional automation solutions. This chapter describes the development of an industrial demonstrator using a control infrastructure called Sequence Planner (SP), and presents some lessons learned during development. SP is based on ROS2 and it is designed to aid in handling the increased complexity of these new systems using formal models and online planning algorithms to coordinate the actions of robots and other devices. During development, SP can auto generate ROS nodes and message types as well as support continuous validation and testing. SP is also designed with the aim to handle traditional challenges of automation software development such as safety, reliability and efficiency. In this chapter, it is argued that ROS2 together with SP could be an enabler of intelligent automation for the next industrial revolution.

1 Introduction

As anyone with experience with real automation development knows, developing and implementing a flexible and robust automation system is not a trivial task. There are currently many automation trends in industry and academia, like Industry 4.0, cyber-physical production systems, internet of things, multi-agent systems and artificial intelligence. These trends try to describe how to implement and reason about flexible and robust automation, but are often quite vague on the specifics. When it comes down to the details, there is no silver bullet [1].

Volvo Group Trucks Operations has defined the following vision to better guide the research and development of next generation automation systems: Future Volvo factories [2], will be *re-configurable* and *self-balancing* to better

handle rapid changes, production disturbances, and diversity of the product. Collaborative robots and other machines can support operators at workstations, where they use the same smart tools and are interchangeable wit operators when it comes to performing routine tasks. Operators are supported with mixed reality technology, and are provided with digital work instructions and 3D geometry while interacting intuitively with robots and systems. Workstations are equipped with smart sensors and cameras that feed the system with real-time status of products, humans, and other resources in the environment. Moreover, they are supported by advanced yet intuitive control, dynamic safety, online optimization and learning algorithms.

Many research initiatives in academia and industry have tried to introduce collaborative robots ("cobots") in the final assembly [3]–[5]. Despite their relative advantages, namely that they are sometimes cheaper and easier to program and teach [5] compared to conventional industrial robots, they are mostly deployed as robots "without fences" for co-active tasks [6]. Current cobot installations are in most cases not as flexible, robust or scalable as required by many tasks in manual assembly. Combined with the lack of industrial preparation processes for these types of systems, new methods and technologies must be developed to better support the imminent industrial challenges [7].

The Robot Operating System and Sequence Planner

This chapter discusses some of the challenges of developing and implementing an industrial demonstrator that includes robots, machines, smart tools, human-machine interfaces, online path and task planners, vision systems, safety sensors, etc. Coordination and integration of the aforementioned functionality requires a well-defined communication interface with good monitoring properties.

During the past decade, various platforms have emerged as middle-ware solutions trying to provide a common ground for integration and communication. One of them is the Robot Operating System (ROS), which stands out with a large and enthusiastic community. ROS enables all users to leverage an ever-increasing number of algorithms and simulation packages, as well as providing a common ground for testing and virtual commissioning. ROS2 takes this concept one step further, integrating the scalable, robust and well-proven Data Distribution Service (DDS) as its communications layer, eliminating

many of the shortcomings of ROS1.

ROS2 systems are composed of a set of nodes that communicate by sending typed messages over named topics using a publish/subscribe mechanism. This enables a quick and semi-standardized way to introduce new drivers and algorithms into a system. However, having reliable communication, monitoring tools, as well as a plethora of drivers and algorithms ready to be deployed is not enough to be able to perform general automation. When composing a system of heterogeneous ROS2 nodes, care needs to be taken to understand the behavior of each node. While the interfaces are clearly separated into message types on named topics, knowledge about the workings of each node is not readily available. This is especially true when nodes contain an internal state that is not visible to the outside world. In order to be able to coordinate different ROS2 nodes, a control system needs to know both *how to control* the nodes making up the system, as well as how these nodes *behave*.

To control the behavior of nodes, an open-source control infrastructure called Sequence Planner (SP) has been developed in the last years. SP is used for controlling and monitoring complex and intelligent automation systems by keeping track of the complete system state, automatically planning and re-planning all actions as well as handling failures or re-configurations.

This chapter presents the development of a flexible automation control system using SP and ROS2 in a transformed truck engine final assembly station inspired by the Volvo vision. The chapter contributes with practical development guidelines based on the **lessons learned** during development, as well as detailed design rationales from the final demonstrator, using SP built on top of ROS2.

The next section introduces the industrial demonstrator that will be used as an example throughout the chapter. Section 3 discusses robust discrete control, why distributed control states should be avoided and the good practice of using state-based commands. The open-source control infrastructure SP is introduced in Section 4 and the generation of ROS2 code for logic and tests is presented in Section 5.

2 An industrial demonstrator

The demonstrator presented in this paper is the result of a transformation of an existing manual assembly station from a truck engine final assembly line, shown in Figure 1, into an intelligent and collaborative robot assembly station, shown in Figure 2.

Figure 1: The original manual assembly station.

In the demonstrator, diesel engines are transported from station to station in a predetermined time slot on Automated Guided Vehicles (AGVs). Material to be mounted on a specific engine is loaded by an operator from kitting facades located adjacent to the line. An autonomous mobile platform (MiR100) carries the kitted material to be mounted on the engine, to the collaborative robot assembly station.

In the station, a robot and an operator work together to mount parts on the engine by using different tools suspended from the ceiling. A dedicated camera system keeps track of operators, ensuring safe coexistence with machines. The camera system can also be used for gesture recognition.

Before the collaborative mode of the system starts, an authorized operator has to be verified by a RFID tag. After verification, the operator is greeted by the station and operator instructions are shown on a screen. If no operator is verified, some operations can still be executed independently by the robot, however, violation of safety zones triggers a safeguard stop.

Figure 2: Collaborative robot assembly station controlled by a network of ROS2 nodes. A video clip: `https://youtu.be/TK1Mb38xiQ8`.

After the AGV and the kitting material have arrived, a Universal Robots (UR10) robot and an operator lift a heavy ladder frame on to the engine. After placing the ladder frame on the engine, the operator informs the control system with a button press on a smartwatch or with a gesture, after which the UR10 leaves the current end-effector and attaches to the nutrunner used for tightening bolts. During this tool change, the operator starts to place 24 bolts, which the UR10 will tighten with the nutrunner.

During the tightening of the bolts, the operator can mount three oil filters. If the robot finishes the tightening operation first, it leaves the nutrunner in a floating position above the engine and waits for the operator. When the operator is done, the robot attaches a third end-effector and starts performing the oil filter tightening operations. During the same time, the operator attaches two oil transport pipes on the engine, and uses the same nutrunner to tighten plates that hold the pipes to the engine. After executing these operations, the AGV with the assembled engine and the empty MiR100 leave the collaborative robot assembly station. All of this is controlled by the hierarchical control infrastructure Sequence Planner that uses ROS2.

2.1 ROS2 structure

ROS2 has a much improved transport layer compared to ROS1, however, ROS1 is still ahead of ROS2 when it comes to the number of packages and active developers. In order to embrace the strengths of both ROS1 and ROS2, i.e. to have an extensive set of developed robotics software and a robust way to communicate, all sub-systems in the demonstrator communicate over ROS2 where a number of nodes have their own dedicated ROS1 master behind a bridge [8].

A set of related nodes located on the same computer are called a hub [8], where one or more hubs can be present on the same computer. ROS hubs are considered to have an accompanying set of bridging nodes that pass messages between ROS and ROS2.

In the demonstrator, ROS nodes are distributed on several computers shown in Table 1, including standard PCs as well as multiple Raspberry Pies and a Lattepanda Alpha (LP Alpha). Most of the nodes are only running ROS2.

No.	Name	ROS v.	Computer	OS	Arch.	Explanation
1	Tool ECU	Dashing	Rasp. Pi	Ubuntu 18	ARM	Smart tool and lifting system control
2	RSP ECU	Dashing	Rasp. Pi	Ubuntu 18	ARM	Pneumatic conn. control and tool state
3	Dock ECU	Dashing	Rasp. Pi	Ubuntu 18	ARM	State of docked end-effectors
4	MiRCOM	Dashing	LP Alpha	Ubuntu 18	amd64	ROS2 to/from REST
5	MiR	Kinetic+Dashing	Intel NUC	Ubuntu 16	amd64	Out-of-the-box MiR100 ROS Suite
6	RFIDCAM	Dashing	Desktop	Win 10	amd64	Published RFID and Camera data
7	UR10	Kinetic+Dashing	Desktop	Ubuntu 16	amd64	UR10 ROS Suite
8	DECS	Dashing	Laptop	Ubuntu 18	amd64	Sequence Planner

Table 1: Overview of the computers in the demonstrator. The ROS versions used were Kinetic for ROS1 and Dashing for ROS2. Some nodes, like number 5 and 7, need both.

All computers in the system are communicating using ROS2 and they can easily be added or removed. All computers connect to the same VPN network as well, to simplify the communication over cellular 4G.

One important **lesson learned** was that ROS1 nodes should only communicate with a ROS1 master on the same computer, else the overall system becomes hard to setup and maintain. During the demonstrator development, we tried to use ROS2 nodes whenever possible, and we only used ROS2 for communication between computers.

There is a number of available implementations of DDS, and since DDS is standardized, ROS2 users can choose an implementation from a vendor that

suits their needs. This is done through a ROS Middleware Interface (RMW), which also exposes Quality of Service (QoS) policies.

QoS policies allow ROS2 users to adjust data transfer in order to meet desired communication requirements. Some of these QoS policies include setting message history and reliability parameters. This QoS feature is crucial, especially in distributed and heterogeneous systems, since setups are usually unique.

However, in the demonstrator described in this chapter, default settings were sufficient for all nodes since there were no real-time requirements. Even though standard QoS settings were used, how the messaging and communication was setup highly influenced the robustness of the system. In the next section, we will study this in more details. The communication between the hubs and the control system is in some cases auto-generated by Sequence Planner and will be introduced in more details in Section 4.

The main focus when developing this demonstrator was to perform the engine assembly using both humans and intelligent machines in a robust way. Other important aspects are non-functional requirements like safety, real-time constraints and performance. However, these challenges were not the main focus when developing the demonstrator. This will be important in future work, especially to handle the safety of operators.

3 Robust discrete control

The presented demonstrator consists of multiple sub-systems that need to be coordinated and controlled in a robust way. A robust discrete control system must handle unplanned situations, restart or failures of sub-systems, as well as instructing and monitoring human behavior.

During development, we also **learned** that one of the most complex challenges for a robust discrete control system, is how to handle asynchronous and non-consistent control states. In this section, we will therefore take a look at this and why it is important to avoid having a distributed control state and why state-based commands should be used instead of event-based. In many implementations, this challenge is often neglected and handled in an ad-hoc fashion. To better understand this challenge and how to handle it, let us start with some definitions.

3.1 States and state variables

There are many different ways to define the state of a system. For example, if we would like to model the state of a door, we can say that it can be: $\{opened, closed\}$, or maybe even: $\{opening, opened, closing, closed\}$. We could also use the door's angle, or maybe just a boolean variable. Depending on what we would like to do with the model and what can we actually measure, we will end up with very different definitions. In this chapter, a model of a state of a system is defined using state variables:

Definition 1: *A state variable v has a domain $V = \{x_1, \ldots, x_n\}$, where each element in V is a possible value of v. There are three types of variables: v^m:* measured *state variables, v^e:* estimated *state variables, and v^c:* command *state variables. For simplicity, we will use the following notation when defining a variable: $v = \{x_1, \ldots, x_n\}$.*

State variables are used when describing the state of a system, where a state is defined as follows:

Definition 2: *A state S is a set of tuples $S = \{\langle v_i, x_i\rangle, \ldots, \langle v_j, x_j\rangle\}$, where v_i is a state variable and $x_i \in V_i$ is the current value of the state variable. Each state variable can only have one current value at the time.*

Let us go back to the door example to better understand states and state variables. The door can for example have the following state variables:

$$pos^e = \{opened, closed\}$$

$$locked^m = \{true, false\}$$

In this example, the position of the door is either *opened* or *closed* and is defined by the *estimated* state variable pos^e. It is called estimated since we can not measure the position of this example door. The pos^e variable must therefore be updated by the control system based on what actions have been executed (e.g. after we have opened the door, we can assume that it is opened).

The door also has a lock, that can either be locked or not, and is defined by the *measured* state variable $locked^m$. It is called measured since this door has a sensor that tells us if the door is locked or not.

The door can be in multiple states, for example, the state $\{\langle pos^e, closed\rangle, \langle locked^m, true\rangle\}$ defines that the door is closed and locked. To be able to change the state of the door, a control system needs to perform actions that unlocks and opens the door.

3.2 Event-based commands and actions

When controlling different resources in an automation system, a common control approach is for the control system to send commands to resources, telling them what to do or react on. Also, the control system reacts on various events from the resources. When communicating in this fashion, which we can call event-based communication, all nodes will wait for and react on events. This type of communication and control often leads to a highly reactive system but is often challenging to make robust.

In ROS2, nodes communicate via topics, services or actions. It is possible to get some communication guarantees from the QoS settings in DDS, however, even if the communication is robust, we **learned** the hard way that it is not easy to create a robust discrete control system using only event-based communication. To better explain this, let us study how one of the nodes from the demonstrator could have been implemented, if we have had used event-based communication:

EX. **The nutrunner: event-based communication**

In the demonstrator, either the robot or the human can use the nutrunner. When started, it runs until a specific torque or a timeout has been reached. In the example, the nutrunner is controlled by the following commands: *start* or *reset*, and responds with the events *done* or *failed*. The communication could be implemented with ROS actions but for simplicity, let us use two publish-subscribe topics and the following ROS messages:

```
# ROS2 topic:  /nutrunner/command
string cmd     # 'start', or 'reset'

# ROS2 topic:  /nutrunner/response
string event   # 'done', or 'failed'
```

To start the nutrunner, the *start* command message is sent to the nutrunner node, which will start running. When a specific torque is reached, the node sends a *done* event back. If for some reason the torque is not reached after a timeout, or if the nutrunner fails in other ways, it will respond with a *failed* event. To be able to start the nutrunner again, the *reset* command must be sent to initialize the nutrunner.

If we can guarantee that each message is only received once and everything behaves as expected, this may work quite fine. However, if for some reason either the controller or the node fails and restarts, we will have a problem. If the controller executes a sequence of actions, it will assume that it should first send the *start* command and then wait for *done* or *failure*. However, if the nutrunner node is in the wrong state, for example if it is waiting for a *reset* command, nothing will happen. Then the control sequence is not really working, since the states of the two nodes are out of sync (i.e. the control node thinks that the nutrunner node is in a state where the control can send start).

A common problem in many industrial automation systems is that the state of the control system gets out of sync with the actual states of some resources. The reason for this is that many implementations are based on sequential control with event-based communication to execute operations, with the assumption that the sequence always starts in a well defined global state. However, this is often not the case, e.g. when a system is restarted after a break or a failure.

During development, we **learned** that in order to achieve flexibility and to handle failures and restart, a system should never be controlled using strict sequences or if-then-else programming. When the control system gets out of sync, the common industrial practice is to remove all products and material from the system, restart all resources and put them in well defined initial "home" states. Since we need a robust control that can handle resource failures, flexible and changing work orders and other unplanned events, a better control strategy is needed.

So, if the problem is that some nodes need to know the states of other nodes, maybe we can solve this by sharing internal state of the node and keep the event-based communication?

3.3 Distributed state

If each node shares its state and is responsible for updating it, we can say that the global state and the control is distributed on all nodes. This is a popular control strategy in many new automation trends, since it is simple to implement and hide some of the node details from other nodes. Let us take a look at how a distributed control strategy could have been implemented in the nutrunner example:

EX. **The nutrunner: Sharing state**

In this example, the node communicates with the hardware via 1 digital output and 3 digital inputs. These can be defined as state variables:

$$run^c = \{false, true\}$$

$$run^m = \{false, true\}$$

$$tqr^m = \{false, true\}$$

$$fail^m = \{false, true\}$$

To run the nutrunner, the command state variable run^c is set to true, which is a digital output, and then the nutrunner responds back by setting the measured state variable run^m to true when it has started. When the torque has been reached, tqr^m becomes true, or if it fails or timeouts, $fail^m$ becomes true. These inputs are reset to false by setting run^c to false again, which can also be used during execution to stop the nutrunner.

Since the nutrunner node is controlled via events (like in the example before), it does not need to expose these details, but instead implements this as an internal node control loop. In our example it communicates its state using the following message:

```
# ROS2 topic:  /nutrunner/state
string state   #'idle', 'running', 'torque', or 'failed'
```

Now, when the nutrunner node is sharing its state, it is easier for an external control system to know the correct state and act accordingly.

However, it is quite challenging to implement a robust state machine that actually works as expected.

When hiding implementation details by using a distributed control state, like in the nutrunner example, the control will initially appear to be simple to implement and may work at first. However, based on our struggling during the demonstrator development, we found out that it is still quite hard to implement fully correct logic in each node and keep the states of all nodes in sync with each other. The main reason for this is that the control system anyway needs to know the details of each node to actually figure out how to restart a system when something happens. So, the **lesson learned:** restart of most nodes almost always depends on the state of other nodes.

It is also challenging to handle safety concerns where the system must guarantee some global specifications when the detailed control is distributed. We have learned the hard way that a local control loop and internal state in each node almost always becomes tangled with multiple cross-cutting concerns making the complete system hard to maintain and troubleshoot. Another approach is to centralize the control and to use state-based commands.

3.4 State-based commands

The benefit of centralized control is that the control system knows about everything and can guarantee global requirements. It can figure out the optimal operation sequence and can restart the complete system in an efficient way.

In the door control earlier, the controller sent the *open* event and then it was waiting for the *done* event. When using state-based commands, the controller instead sends a reference position, $refpos^c$, telling the door what state it wants it to be in while the door is sending back its real state. These messages are sent when something changes as well as at a specific frequency. If one of the commands is lost, another one will soon be sent.

Let us look at how a state-based command could be implemented for the nutrunner example.

EX. **The nutrunner: State-based communication**

Instead of using events to control the nutrunner, the control system and the node instead communicate using the already defined state variables

with the following messages:

```
# ROS2 topic:  /nutrunner/command
bool run_c

# ROS2 topic:  /nutrunner/state
bool run_m
bool tqr_m
bool fail_m
bool run_c
```

The control system is sending a reference state while the nutrunner is returning its current state as well as mirroring back the reference. The centralized control system now knows everything about the nutrunner since in practice, the detailed control loop is moved from the nutrunner node to the control node. The mirror is monitored by an external ROS2 node which checks that all nodes are receiving their commands.

During the development of the demonstrator, we discovered that by using state-based commands and avoiding local states in each node, it is possible to implement control strategies for efficient restart, error handling, monitoring and node implementation. It is much easier to recover the system by just restarting it, since the nodes do not contain any local state. Stateless nodes also makes it possible to add, remove or change nodes while running, without complex logic. This has probably been one of the most important **lessons learned** during the development.

The control of the demonstrator is using centralized control, a global state, and state-based communication. While state-based communication requires a higher number of messages to support control compared to event-based communication, and centralized control with a global state is harder to scale compared to decentralized approaches, we believe that these trade-offs are worth it when creating a robust and efficient control architecture for these types of automation systems. Perhaps the bigger challenge with this approach though, is that the control system needs to handle all the complexity to make it robust. To aid in handling this complexity, we have developed a new control infrastructure called Sequence Planner for robust automation of many sub-systems.

4 Sequence Planner

The use of state-based commands introduced in Section 3 implies that a number of devices should continuously get a reference (or goal) state. An overall control scheme is then needed in order to choose what these goal states should be at all time. At the same time, the control system should react to external inputs like state changes or events from machines, operators, sensors, or cameras. Developing such systems quickly becomes difficult due to all unforeseen situations one may end up in. Manual programming becomes too complex and executing control sequences calculated off-line become very difficult to recover from when something goes wrong.

Several frameworks in the ROS community are helping the user with composing and executing robot tasks (or algorithms). For example, the framework ROSPlan [9] that uses PDDL-based models for automated task planning and dispatching, SkiROS [10] that simplifies the planning with the use of a skill-based ontology, eTaSL/eTC [11] that defines a constraint-based task specification language for both discrete and continuous control tasks or CoSTAR [12] that uses Behavior Trees for defining the tasks. However, these frameworks are mainly focused on robotics rather than automation.

4.1 A new control infrastructure

Based on what we **learned**, we have employed a combination of on-line automated planning and formal verification, in order to ease both modeling and control. Formal verification can help us when creating a model of the correct behavior of device nodes while automated planning lets us avoid hard-coded sequences and "if-then-else" programming and allowing us to be resource-agnostic on a higher level. We end up with a control scheme that continuously deliberates the best goals to distribute to the devices. This is implemented in our research software Sequence Planner (SP).

SP is a tool for modeling and analyzing automation systems. Initially [13], the focus was on supporting engineers in developing control code for programmable logical controllers (PLCs). During the first years, algorithms to handle product and automation system interaction [14], and to visualize complex operation sequences using multiple projections [15] were developed. Over the years, other aspects have been integrated, like formal verification and synthesis using *Supremica* [16], restart support [17], cycle time optimiza-

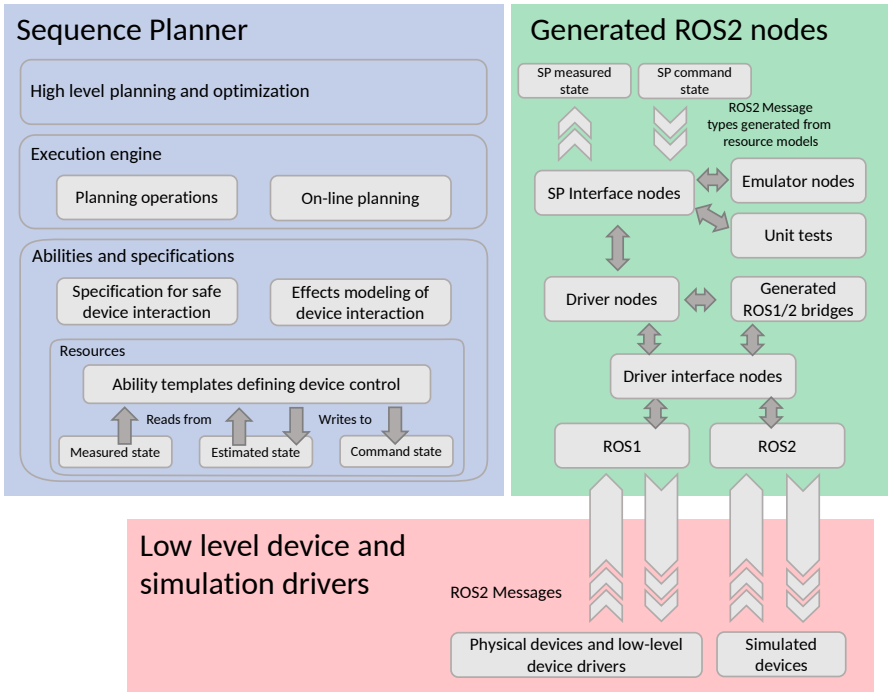


Figure 3: Sequence Planner control infrastructure overview.

tion [18], energy optimization and hybrid systems [19], online monitoring and control [20], as well as emergency department online planning support [21]. Recently, effort has been spent to use the developed algorithms and visualization techniques for control and coordination of ROS- and ROS2-based systems [22]. This section will give an overview of how SP can be used to develop ROS2-based automation systems.

The remainder of this section will describe how these SP control models look like, starting from the bottom up of the left part of Fig. 3. In Section 5 it is described how the model is translated into ROS2 nodes with corresponding messages (top right of Fig. 3. Because model-based control is used, it is essential that the models accurately represent the underlying systems (bottom of Fig. 3). Therefore Section 5.2 describes how randomized testing is used to ease the development of ROS2 nodes.

4.2 Resources

Devices in the system are modeled as *resources*, which groups the device's local state and discrete descriptions of the tasks the device can perform. Recall Definition 1, which divides system state into variables of three kinds: *measured state*, *command state*, and *estimated state*.

Definition 3: *A resource i is defined as $r_i = \langle V_i^m, V_i^c, V_i^e, O_i \rangle, r_i \in R$ where V_i^m is a set of* measured *state variables, V_i^c is a set of* command *state variables, V_i^e is a set of* estimated *state variables, and O_i is a set of* generalized operations *defining the resource's abilities.*

R is the set of all resources in the system.

The resources defined here are eventually used to generate ROS2 nodes and message definition files corresponding to state variables.

4.3 Generalized operations

Control of an automation system can be abstracted into performing *operations*. By constructing a transition system modeling how operations modify states of the resources in a system, formal techniques for verification and planning can be applied. To do this in a manner suitable to express both low-level ability operations and high-level planning operations, we define a *generalized operation*.

Definition 4: *A generalized operation j operating on the state of a resource i is defined as $o_j = \langle P_j, G_j, T_j^d, T_j^a, T_j^E \rangle, o_j \in O_i$. P_j is a set of named predicates over the state of resource variables V_i. G_j is a set of un-named guard predicates over the state of V_i. Sets T^d and T^a define* control *transitions that update V_i, where T^d defines transitions that require (external from the ability) deliberation and T^a define transitions that are automatically applied whenever possible. T_j^E is a set of* effect *transitions describing the possible consequences to V_i^M of being in certain states. A transition $t_i \in \{T^d \cup T^a \cup T^E\}$ has a guard predicate which can contain elements from P_j and G_j and a set of actions that update the current state if and only if the corresponding guard predicate evaluates to true.*

T_j^d, T_j^a, and T_j^E have the same formal semantics, but are separated due to their different uses. The effect transitions T_j^E define how the *measured state* is updated, and as such they are not used during actual low-level control like the control transitions T_j^d and T_j^a. They are important to keep track of since

they are needed for on-line planning and formal verification algorithms, as well as for simulation based validation.

It is natural to define when to take certain actions in terms of what state the resource is currently in. To ease both modeling, planning algorithms and later on online monitoring, the guard predicates of the generalized operations are separated into one set of named (P_j) and one set of un-named (G_j) predicates. The named predicates can be used to define in what state the operation currently is in, in terms of the set of local resource states defined by this predicate. The un-named predicates are used later in Section 4.6 where the name of the state does not matter.

4.4 Ability operations

The behavior of resources in the system is modeled by *ability operations* (abilities for short). While it is possible to define transitions using only un-named guard predicates in G (from Definition 4), it is useful to define a number of "standard" predicate names for an ability to ease modeling, reuse and support for online monitoring. In this work, common meaning is introduced for the following predicates: *enabled* (ability can start), *starting* (ability is starting, e.g. a handshaking state), *executing* (ability is executing, e.g. waiting to be finished), *finished* (ability has finished), and *resetting* (transitioning from finished back to enabled). In the general case, the transition between *enabled* and *starting* has an action in T^d, while the transition from *finished* has an action in T^a. In other types of systems, other "standard" names could be used (e.g. *request* and *receive*).

EX. **The nutrunner: resource and ability template**

The resource nr containing state variables of the nutrunner can be defined as $r_{nr} = \langle \{run^m, tqr^m, fail^m\}, \{run^c\}, \emptyset, O_{nr} \rangle$. Notice that $V^e_{nr} = \emptyset$. This is the ideal case, because it means all local state of this resource can be measured.

The table below shows the transitions of a "run nut" ability, where each line makes up one possible transition of the ability. The ability models the task of running a nut, by starting to run the motors forward until a pre-programmed torque (tqr^m) has been reached. Notice that for

this ability, $G = \emptyset$. Control actions in T^d are marked by $\bigstar$.

pred. name	predicate	control actions	effects
enabled	$\neg run^c \wedge \neg run^m$	$run^c = T^{\bigstar}$	-
starting	$run^c \wedge \neg run^m$	-	$run^m = T$
executing	$run^c \wedge run^m \wedge \neg tqr^m$	-	$tqr^m = T \vee fail^m = T$
finished	$run^c \wedge run^m \wedge tqr^m$	$run^c = F$	-
failed	$run^c \wedge run^m \wedge fail^m$	$run^c = F$	-
resetting	$\neg run^c \wedge run^m$	-	$run^m = F$, $tqr^m = F$, $fail^m = F$

To implement this logic, we have **learned** that it is as complex as implementing it in a distributed state machine, which we talked about in Section 3. However, since this model is directly verified and tested together with the complete system behavior, we directly find the errors and bugs. This is much harder in a distributed and asynchronous setup.

While abilities make for well isolated and reusable components from which larger systems can be assembled, they can mainly be used for manual or open loop control. The next step is therefore to model *interaction* between resources, for example between the robot and the nutrunner, or between the state of the bolts and the nutrunner.

4.5 Modeling resource interaction

Fig. 3 illustrates two types of interaction between resources: "specification for safe device interaction" and "effects modeling of device interaction". The latter means modeling what can happen when two or more devices interact. As it might not be possible to *measure* these effects, many of them will be modeled as control actions updating estimated state variables.

Given a set of resources and generalized operations as defined by Definition 4, a complete control model is created by instantiating the needed operations into a *global* resource r_g ($r_g \notin R$). Additional estimated state added if needed in order to express the result of different resources interacting with each other. See Example 4.2.

Safety specifications are created to ensure that the resources can never do something "bad". The instantiated ability operations can be used together with a set of global specifications to formulate a supervisor synthesis problem from the variables and transitions in r_g. Using the method described in [23],

the solution to this synthesis problem can be obtained as additional guard predicates on the deliberate transitions in T^d. Examples of this modeling technique can be found in [24], [25]. By keeping specifications as a part of the model, we **learned** that there are fewer points of change which makes for faster and less error-prone development compared to changing the guard expressions manually.

4.6 Planning operations

While ability operations define low-level tasks that different devices can perform, planning operations model how to make the system do something *useful*. As the name suggests, planning operations define intent in the form goal states.

A planning operation has a preconditions which define when it makes sense to try to reach the goal, and a postcondition specifying a target, or goal, state. Planning operations also introduce an estimated state variable o^e to keep track of its own state, $o^e = \{initial, executing, finished\}$. When in the initial state, if the precondition is satisfied, the planning operation updates its estimated state variable to *executing* which will trigger this planning operation to be considered by the on-line planner. When the goal is reached, the operation's state variable transitions to *finished*, and the goal is removed from the on-line planner.

EX. **The nutrunner: Specifying device interaction and planning operations**

We would like to create a planning operation called `TightenBolt1`. When a bolt is in position, which is a precondition of `TightenBolt1`, the operation instructs the system to reach the goal where bolt1 has been tightened. This will happen when the nutrunner has reached the correct torque and the robot is in the correct position at the bolt.

The robot has a number of state variables and in this example, we are interested in its position, where the *measured* state variable $ur.pos^m = \{pos_1, \ldots, pos_n\}$ defines what named pose the robot can be in. In our system we would also like to know the state of bolt1, which is modeled with the *estimated* state variable $bp_1^e = \{empty, placed, tightened\}$. This state variable is not only used by one ability, but in multiple abilities and

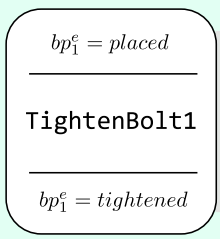
planning operations. If a variable is only used by planning operations, it is updated by their actions when starting or when reaching the goal, but in this case also other abilities need to know the state of the bolts.

For the planner to be able to find a sequence of abilities that can reach the goal, an ability needs to take the action bp_1^e = tightened. The nut running ability or the robot can be extended with this extra action, but then special variants of abilities need to be made. If, for example, we extend the nut running ability, it needs to be duplicated for each bolt with just minor differences.

A better approach is to generate new special abilities to track these results. In this case, an ability is created only with the following transition:

pred. name	predicate	control actions	effects
-	$tqr^m \wedge ur.pos^m = bp_1$	$bp_1^e = tightened$	-

A natural way to model `TightenBolt1`, is to start in the pre-state defined by its precondition $bp_1^e = placed$ and end in the post-state defined by the postcondition $bp_1^e = tightened$. The figure below shows a planning operation, with a sequence of dynamically matched ability operations (to the right) to reach the goal state $bp_1^e = tightened$.




```
1.ur.gotoPreAttachOFToolFar
2.ur.gotoHome
3.ur.gotoAboveEngine
4.ur.gotoFarAboveBoltPair1
5.ur.gotoCloseAboveBoltPair1
6.nr.runNut1
7.ur.gotoAtBoltPair1
```

Modeling operations in this way does two things. First, it makes it possible to add and remove resources from the system more easily - as long as the desired goals can be reached, the upper layers of the control system do not need to be changed. Second, it makes it easier to model on a high level, eliminating the need to care about specific sequences of abilities. As shown in Example 4.2, the operation `TightenBolt1` involves sequencing several abilities controlling the robot and the tool to achieve the goal state of the operation.

4.7 Execution engine

The execution engine of the control system keeps track of the current state of all variables, planning operations, ability operations and two deliberation plans. The execution engine consists of two stages, the first evaluates and triggers planning operation transitions and the second evaluates and triggers ability operation transitions. When a transition is evaluated to true in the current state and is enabled, the transition is triggered and the state is updated. After the two steps have been executed, updated resource state is sent out as ROS2 messages.

Each stage includes both deliberation transitions and automatic transitions. The automatic transitions will always trigger when their guard predicate is true in the current state, but the deliberation transitions must also be enabled by the deliberation plan. The deliberation plan is a sequence of transitions defining the execution order of deliberation transitions. The plan for the planning operation stage is defined either by a manual sequence or by a planner or optimizer on a higher level. For the ability stage, the deliberation plan is continuously refined by an automated planner that finds the best sequence of abilities to reach a goal defined by the planning operations.

In this work, planning is done by finding counter-examples using bounded model checking (BMC) [26]. Today's SAT-based model checkers are very efficient in finding counter-examples, even for systems with hundreds of variables. By starting in the current state, the model of the system is unrolled to a SAT problem of increasing bound, essentially becoming an efficient breadth-first search. As such, a counter-example (or deliberation plan) is minimal in the number of transitions taken from the current state. Additionally, well-known and powerful specification languages like *Linear Temporal Logic* (LTL) [27] can be used to formulate constraints.

When decisions are taken by the SP execution engine instead of by the resources themselves, nodes essentially become shallow wrappers around the actual devices in the system. The fact that the core device behavior is modeled in SP allows for generation of a lot boilerplate code (and tests!) in the nodes, which is described in the next section.

5 Auto generated ROS nodes

The architecture of the system is based on the ROS hubs described in Section 2.1 and in [8], which allow us to separate the system into functional entities. Each ROS hub represents one SP resource which is exchanging messages based on the SP model of the resource. Based on these models of resources, SP's component generator module generates a set of ROS2 nodes with accompanying messages during compile-time.

5.1 Auto generated code for resource

Five nodes per resource hub are generated: *interfacer*, *emulator*, *simulator*, *driver* and *test*. The *interfacer* node serves as a standardized interface node between SP and nodes of the resource. *Emulator* nodes are generated based on abilities defined in SP and are used when testing the control system since they emulate the expected behavior of the ability. *Simulator* and *driver* nodes implement the actual device control and also an initial template to be used when implementing the connection with the real drivers in ROS. *Test* nodes implement automatic testing of *simulator* and *driver* nodes based on the SP model.

To use the abilities in formal planning algorithms and to support testing of the SP model without being connected to the real subsystem, *measured* variables must be updated by *someone*. We have chosen to place this updating as individual *emulation* nodes [28]rather than to let SP perform it internally. This allows us to always maintain the structure of the network which allows an easy transition to simulated or actual nodes at any time, while at the same time keeping the core execution engine uncluttered. The internal structure of the emulator node is quite simple and can be fully generated based on the SP model. This is because the internal state of an emulation does not have to reflect the internal state of the target which it is emulating, it only needs to mimic the observable behavior to match an existing target.

After the SP model has been tested with generated emulator nodes, the next step in the workflow is to use a variety of simulators instead of emulators connected in the ROS2 network via *simulator* nodes. These nodes are partially generated based on the model of the resource and partially manually assembled. The manually assembled part is decoupled from the main node and is imported into the *simulator* so that the node itself can be independently

re-generated when the SP model changes.

These imports contain interface definitions for specific hardware and software. In the example of the nutrunner, the *driver* node is run on a Raspberry Pi and its manually written part contains mapping between variables and physical inputs and outputs.

Per resource, the *Simulator* node talks to SP over the *interfacer* node using same message types generated for all three nodes, *emulator*, *simulator* and *driver*. This way, the *interfacer* node does not care if the equipment is real, simulated or just emulated.

The driver nodes are in most cases exactly the same as simulator nodes, however we distinguish them by trying to be general. Both the simulator and the driver nodes use same imported manually assembled interfacing components.

EX. **The nutrunner: model-based component generation**

Given the nutrunner's "run nut" ability defined in Example 4.1, the component generator module generates one ROS2 package containing the different node types for that resource and another package containing the necessary message types:

```
├── nutrunner
│   ├── launch
│   ├── package.xml
│   ├── resource
│   │   └── nutrunner
│   ├── setup.cfg
│   ├── setup.py
│   ├── src
│   │   ├── __init__.py
│   │   ├── nutrunner_sp_driver.py
│   │   ├── nutrunner_sp_emulator.py
│   │   ├── nutrunner_sp_interfacer.py
│   │   ├── nutrunner_sp_simulator.py
│   │   └── nutrunner_sp_test.py
│   └── test
│       ├── test_copyright.py
│       ├── test_flake8.py
│       └── test_pep257.py
├── nutrunner_msgs
│   ├── bridges
│   ├── CMakeLists.txt
│   ├── launch
│   ├── mapping_rules
│   │   ├── nutrunner_12.yaml
│   │   └── nutrunner_21.yaml
│   ├── msg
│   │   ├── NutrunnerDriverToInterfacer.msg
│   │   ├── NutrunnerInterfacerToDriver.msg
│   │   ├── NutrunnerInterfacerToSP.msg
│   │   └── NutrunnerSPToInterfacer.msg
│   └── package.xml
```

5.2 Model based testing

Unit-testing is a natural part of modern software development. In addition to the nodes generated from SP models, *tests* are also generated. Because the SP model clearly specifies the intended behavior of the nodes via *effects*, it is also possible to generate tests which can be used to determine if the model and the underlying driver implementation do in fact share the same behavior. The generated tests are run over the ROS2 network, using the same interfaces as the control system implementation. This enables *model based testing* using a seamless mixture of emulated, simulated, and real device drivers, which can be configured depending on what is tested. Based on what we **learned**, the generated unit tests can:

- ensure that the device drivers and simulated devices adhere to the behavior specified in the SP model.
- help eliminate "simple" programming errors that waste development resources, for example making sure that the correct topics and message types are used for user-written code.
- provide means to validate if the specifications in the SP model make sense. While formal methods can guarantee correctness w.r.t. some specification, writing the specification is still difficult and needs to be supported by continuous testing.

The tests generated by SP employ property-based testing using the Hypothesis library [29], which builds on ideas from the original QuickCheck [30] implementation. These tools generate random test vectors to see if they can break user-specified properties. If so, they automatically try to find a minimal length example that violates the given properties. The properties that need to hold in our case are that effects specified in the SP model always need to be fulfilled by the nodes implementing the resource drivers. Of course, when bombarded by arbitrary messages, it is not surprising to also find other errors (e.g. crashes).

EX. **The nutrunner: automated testing during node development**

Let's exemplify with the nutrunner resource again. The figure below shows the invocation of *pytest* for executing a generated unit test. The

test will send arbitrary commands to the nutrunner node, via the same interface that SP uses for control, and check that the responses from the system match what is expected in the SP model.

```
$ pytest nutrunner_sp_tests.py
========================================= test session starts =========================================
platform linux -- Python 3.6.8, pytest-4.6.3, py-1.8.0, pluggy-0.12.0
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/martin/rosbook/dashing_ws/sr
c/nutrunner/src/.hypothesis/examples')
rootdir: /home/martin/rosbook/dashing_ws/src/nutrunner
plugins: hypothesis-4.24.3, repeat-0.8.0, rerunfailures-7.0, cov-2.5.1
collected 1 item

nutrunner_sp_tests.py F                                                                      [100%]

============================================== FAILURES ===============================================
_____________________________________ test_nutrunner_random ______________________________________

    @given(cmdMsg)
>   @settings(max_examples=50, deadline=None)
    def test_nutrunner_random(data):

nutrunner_sp_tests.py:26:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

data = {'seq': 0, 'set_run': False}

    @given(cmdMsg)
    @settings(max_examples=50, deadline=None)
    def test_nutrunner_random(data):
        [ ... code removed ... ]
E           AssertionError: Node did not produce desired effect within time limit
E           assert False

nutrunner_sp_tests.py:195: AssertionError
---------------------------------------------- Hypothesis ----------------------------------------------
Falsifying example: test_nutrunner_random(data={'seq': 0, 'set_run': False})
expecting effect: ['tqr_m = False']
Last message seen: nutrunner_msgs.msg.NutrunnerInterfacerToSP(got_set_run=False, run_m=False, tqr_m=True,
fail_m=False)
====================================== 1 failed in 14.36 seconds ======================================
$
```

The test fails, showing the generated test vector for which the effect failed to emerge, as well the message last observed on the ROS2 network. This makes it easier to spot problems which often arise due to the model and the actual device behavior differing. In this case, it turns out that the driver node does not properly reset tqr^m when run^c resets. Investigating how the driver node works yields:

```
msg.tqr_m = GPIO.input(self.gpi5) == 1
```

It can be seen from the snippet above that the driver node simply forwards what the lower-level device emits to some input pin. It turns out that for this particular nutrunner, the torque reached flag is reset only when the output used to set start running forward goes high.

Instead of introducing more state, and with that, complexity to the driver node, the proper fix for this is to go back and change the SP

model to reflect the actual behavior of the node. By moving the torque reset effect from the *resetting* state to the *starting* state of the ability template definition and re-generating our package, the generated test now succeeds:

```
$ pytest nutrunner_sp_tests.py
========================================= test session starts ==========================================
platform linux -- Python 3.6.8, pytest-4.6.3, py-1.8.0, pluggy-0.12.0
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/martin/rosbook/dashing_ws/sr
c/nutrunner/src/.hypothesis/examples')
rootdir: /home/martin/rosbook/dashing_ws/src/nutrunner
plugins: hypothesis-4.24.3, repeat-0.8.0, rerunfailures-7.0, cov-2.5.1
collected 1 item

nutrunner_sp_tests.py .                                                                      [100%]

====================================== 1 passed in 35.07 seconds =======================================
$
```

The nutrunner node should now be safe to include in the automation system.

The unit-test described in Example 5.2 deals only with testing one resource individually: it is not the instantiated abilities but the templates that are tested. This means that additional preconditions (e.g. handling zone booking) are not tested in this step. As such, while the real driver for the nutrunner could, and probably should be subjected to these kinds of tests, it is not safe to do with, for instance, a robot. Triggering random target states (e.g. robot movements) would be dangerous and such tests need to be run in a more controlled manner. Luckily ROS2 makes it simple to run the tests on a simulated robot by simply swapping which node runs.

It can also be interesting to generate tests from the complete SP model. Consider an ability `ur.moveTo` for moving the UR robot between predefined poses, that is instantiated for a number of different target poses, each with a different precondition describing in which configuration of the system the move is valid. If a simulated node triggers a simulated protective stop when collisions are detected (e.g. between collision objects in the MoveIt! framework), it becomes possible to test whether the SP model is correct w.r.t allowed moves (it would not reach the target if protective stop is triggered). In fact, such a test can be used to *find* all the moves that satisfy the stated property, which eases modeling.

5.3 Node management and integrated testing

The first step in the engineering workflow would be testing the model with emulator nodes, iterating the model until the desired behavior is reached. Afterwards, a simulation or real hardware can be interfaced for one of the resources, while the others remain emulated. This way, targeted model properties can be tested.

To extend the usability of code generation, a node manager is generated to launch node groups for different testing and commissioning purposes, supporting testing of targeted model properties.

Testing the model using emulator nodes can be referred to as virtual commissioning (VC) [31]. The purpose of VC is to enable the control software, which controls and coordinates different devices in a production station, to be tested and validated before physical commissioning.

The concept of integrated virtual preparation and commissioning (IVPC) [32] aims to integrate VC into the standard engineering workflow as a continuous support for automation engineers. Exactly this is gained with the described workflow of continuous model improvements, autogeneration of components and testing using ROS2 as a common platform that supports integration of different smart software and hardware components.

So, during the development, we have **learned**, that model-based code generation speeds up development and eliminates a lot of coding related errors since components are generated as a one-to-one match to the model. This enables for continuous model improvements during development based on behavior of the generated emulator node.

6 Conclusions

This chapter has presented the development of a industrial demonstrator and the control infrastructure Sequence Planner, together with some practical development guidelines and lessons learned. The demonstrator includes robots, machines, smart tools, human-machine interfaces, online path and task planners, vision systems, safety sensors, optimization algorithms, etc. Coordination and integration of all these functionality has required the well-defined communication interface provided by ROS2 as well as the robust task planning and control of Sequence Planner.

From a technical perspective, ROS2, is an excellent fit for industry 4.0 sys-

tems as shown in this chapter. However, being a well-structured and reliable communication architecture is just one part of the challenge. As we have shown, various design decisions will greatly influence the end result, for example in choosing how to communicate between nodes or in how the system behavior is modeled.

Since the presented control approach uses state based communication, the nodes are continuously sending out messages. This implies that care needs to be taken to avoid flooding the network. However, during work on the demonstrator, the bottleneck was more related to planning time rather than network capacity. Future work should include studying how many resources can be handled by SP, at what publishing frequency messages can be reliably received and how this is influenced by different QoS settings.

During the development and implementation of the industrial demonstrator, we learned that:

- To be able to restart the control system, it was important to avoid distributed control state, as the restart of a node almost always depends on the states of other nodes .
- To make the intelligent automation robust and functional, it was much easier to have a centralized control approach.
- It was easier to handle failures, troubleshoot, and maintain the system when using state-based commands.
- For the system to be flexible and robust, we had to stay away from hard-coded control sequences and if-then-else programming.
- Online automated planning and optimization is necessary for any type of intelligent automation.
- It is impossible to develop these types of systems without continuous testing and verification.

The major challenges during development were always related to implementation details and we learned the hard way that the devil is in the details. Developing any type of automation systems will never be a simple task, but it is possible to support the creative design process using algorithms. The control infrastructure Sequence Planner is an open source project that can be found here: `http://github.com/sequenceplanner/sp`.

Acknowledgment

This work was funded by Volvo Groups Trucks Operations and the Swedish Innovation Agency Vinnova through the Production 2030 project Unification and the FFI project Unicorn.

References

[1] F. P. Brooks Jr., "No silver bullet: Essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987, ISSN: 0018-9162.

[2] *Volvo GTO Vision*, `https://www.engineering.com/PLMERP/ArticleID/18868/Vision-and-Practice-at-Volvo-Group-GTO-Industry-40-and-PLM-in-Global-Truck-Manufacturing.aspx`, [Online; accessed 14-Apr-2019].

[3] P. Tsarouchi, A.-S. Matthaiakis, S. Makris, and G. Chryssolouris, "On a human-robot collaboration in an assembly cell," *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 6, pp. 580–589, 2017.

[4] Å. Fast-Berglund, F. Palmkvist, P. Nyqvist, S. Ekered, and M. Åkerman, "Evaluating cobots for final assembly," *Procedia CIRP*, vol. 44, pp. 175–180, 2016, 6th CIRP Conference on Assembly Technologies and Systems (CATS), ISSN: 2212-8271.

[5] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human-robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018, ISSN: 0957-4158.

[6] W. He, Z. Li, and C. L. P. Chen, "A survey of human-centered intelligent robots: Issues and challenges," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 602–609, 2017, ISSN: 2329-9266.

[7] A. Hanna, K. Bengtsson, M. Dahl, E. Erős, P. Götvall, and M. Ekström, "Industrial challenges when planning and preparing collaborative and intelligent automation systems for final assembly stations," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 400–406.

[8] E. Erős, M. Dahl, H. Atieh, and K. Bengtsson, "A ros2 based communication architecture for control in collaborative and intelligent automation systems," in *Proceedings of 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2019)*, Jun. 2019.

[9] M. Cashmore, M. Fox, D. Long, *et al.*, "Rosplan: Planning in the robot operating system," in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'15, Jerusalem, Israel: AAAI Press, 2015, pp. 333–341, ISBN: 978-1-57735-731-5.

[10] F. Rovida, M. Crosby, D. Holz, *et al.*, "Skiros—a skill-based robot control platform on top of ros," in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 121–160, ISBN: 978-3-319-54927-9.

[11] E. Aertbeliën and J. De Schutter, "Etasl/etc: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 1540–1546.

[12] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 564–571.

[13] P. Falkman, B. Lennartson, and K. Andersson, "Specification of production systems using PPN and sequential operation charts," in *2007 IEEE International Conference on Automation Science and Engineering*, Sep. 2007, pp. 20–25.

[14] K. Bengtsson, B. Lennartson, and C. Yuan, "The origin of operations: Interactions between the product and the manufacturing automation control system," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 40–45, 2009, 13th IFAC Symposium on Information Control Problems in Manufacturing.

[15] K. Bengtsson, P. Bergagard, C. Thorstensson, *et al.*, "Sequence planning using multiple and coordinated sequences of operations," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 308–319, Apr. 2012.

[16] P. Bergagård and M. Fabian, "Deadlock avoidance for multi-product manufacturing systems modeled as sequences of operations," in *2012 IEEE International Conference on Automation Science and Engineering: Green Automation Toward a Sustainable Society, CASE 2012, Seoul, 20-24 August 2012*, 2012, pp. 515–520.

[17] P. Bergagård, *On restart of automated manufacturing systems*, ser. PhD Thesis at Chalmers University of Technology. 2015.

[18] N. Sundström, O. Wigström, P. Falkman, and B. Lennartson, "Optimization of operation sequences using constraint programming," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 1580–1585, 2012.

[19] S. Riazi, K. Bengtsson, R. Bischoff, A. Aurnhammer, O. Wigström, and B. Lennartson, "Energy and peak-power optimization of existing time-optimal robot trajectories," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, Aug. 2016.

[20] A. Theorin, K. Bengtsson, J. Provost, *et al.*, "An event-driven manufacturing information system architecture for industry 4.0," *International Journal of Production Research*, pp. 1–15, 2016.

[21] K. Bengtsson, E. Blomgren, O. Henriksson, *et al.*, "Emergency department overview - improving the dynamic capabilities using an event-driven information architecture," in *IEEE International Conference on Emerging technologies and factory automation (ETFA)*, 2016.

[22] M. Dahl, E. Erős, A. Hanna, K. Bengtsson, M. Fabian, and P. Falkman, "Control components for collaborative and intelligent automation systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 378–384.

[23] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *Control Syst. Technol. IEEE Trans.*, vol. 20, no. 6, pp. 1421–1435, 2012.

[24] P. Bergagård, P. Falkman, and M. Fabian, "Modeling and automatic calculation of restart states for an industrial windscreen mounting station," *IFAC-Papers On Line*, vol. 48, no. 3, pp. 1030–1036, 2015.

[25] M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, "Automatic modeling and simulation of robot program behavior in integrated virtual preparation and commissioning," *Procedia Manufacturing*, vol. 11 , pp. 284–291, 2017.

[26] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *International conference on tools and algorithms for the construction and analysis of systems*, Springer, 1999, pp. 193–207.

[27] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[28] E. Erős, M. Dahl, A. Hanna, A. Albo, P. Falkman, and K. Bengtsson, "Integrated virtual commissioning of a ros2-based collaborative and intelligent automation system," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 407–413.

[29] D. R. MacIver, *Hypothesis 4.24*, `https://github.com/HypothesisWorks/hypothesis`, 2018.

[30] K. Claessen and J. Hughes, "Quickcheck: A lightweight tool for random testing of haskell programs," in *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '00, New York, NY, USA: ACM, 2000, pp. 268–279, ISBN: 1-58113-202-6.

[31] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[32] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: Supporting formal methods during automation systems development," *IFAC-Papers On Line*, vol. 49, no. 12, pp. 1939–1944, 2016, 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016, ISSN: 2405-8963.

PAPER B

Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system

Endre Erős, Martin Dahl, Atieh Hanna, Anton Albo, Petter Falkman and Kristofer Bengtsson

Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) vol. 24, pp. 407–413, 2019.

DOI: 10.1109/ETFA.2019.8869444

The layout has been revised.

Abstract

In order to adapt to stricter system delivery and integration requirements, virtual commissioning (VC) has become a well adopted practice in industry. VC is getting increasingly integrated into the overall engineering process, where the control software is continuously tested with the virtual plant model. At the same time, collaborative and intelligent automation systems are becoming an important part of modern industries. In these complex systems, humans perform operations together with collaborative robots, intelligent machines and smart tools. However, performing VC of such complex, distributed and heterogeneous systems demands new ways of interfacing different hardware and software components. This paper discusses the requirements, process and results of integrated virtual commissioning of an industrial collaborative and intelligent automation system use-case. Moreover, this industrial use-case illustrates challenges and exemplifies the need to use the next generation Robot Operating System (ROS2) due to its robust communication layer as well as easy integration with smart devices and algorithms.

1 Introduction

Global demand for high quality products with short life-cycles, that are reasonably priced, is growing. Meanwhile, the variability of such products is rapidly increasing to meet demands of a wider range of customers. In order to stay competitive, companies are decreasing innovation cycles and product delivery times [1]. These changes cause a paradigm shift in companies towards more flexibility and reconfigurability to respond quickly and efficiently to changing production requirements and market demands [2].

Volvo, being a global truck manufacturer, experiences these changes [3]. The vision for a Volvo factory of the future is that is must be more flexible in order to handle the challenges imposed by this paradigm shift [3].

One of the adaptation strategies is the inclusion of collaborative [4] and intelligent automation systems in production. The ergonomic environment in

such production systems enables operators to interface the system through a range of smart human-machine interfaces (HMIs). While collaborative robots and machines support operators at workstations, safety is ensured by dedicated camera and sensor systems and on-line path planning [5]. An example implementation of a safety system using a Microsoft Kinect V2 camera is shown in Fig. 1, [6]. Moreover, humans and robots use common smart tools and are interchangeable when it comes to standard tasks [7].

Another strategy is to perform virtual commissioning (VC) [8] using simulated production systems. The purpose of VC is to enable the control software, which controls and coordinates different devices in a production station, to be tested and validated before physical commissioning.

However, a knowledge gap exists between simulation and control system development since they are traditionally decoupled activities in industry [9].

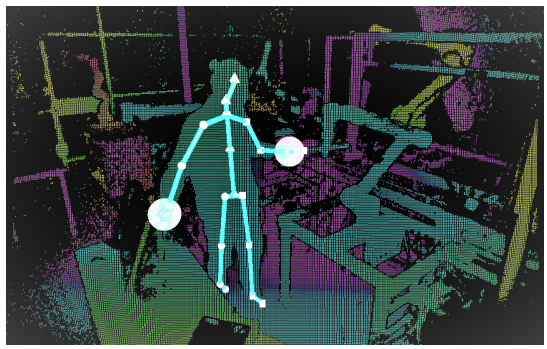


Figure 1: Example of a dedicated operator safety camera feed.

A concept of integrated virtual commissioning (IVC) emerges, which aims to integrate VC into the standard engineering workflow as a continuous support for automation engineers [10]. Additionally, virtual preparation joins this workflow, extending IVC into integrated virtual preparation and commissioning (IVPC) [11].

However, performing IVPC of such complex heterogeneous systems demands a common platform that supports integration of different smart software and hardware components.

In order to ease integration and development of different types of online algorithms for sensing, planning and hardware control, various platforms have

emerged as middle-ware solutions. One platform that stands out is the Robot Operating System (ROS) and its success can be measured by having over 16 million downloads in 2018 [12].

The next generation ROS2 is currently developed, where the developers implemented the Object Management Group standard DDS as the communication middleware considering it to be scalable, robust and well-proven in mission-critical systems. As a result, DDS enables large scale distributed control architectures and implementation of ROS in real-world industrial automation systems.

In the industrial use-case presented in this paper, the forementioned gap in the engineering workflow is bridged using ROS2 as a communication middleware and a platform for continuous integration and testing of simulation, visualization and control.

This paper explains the tools and methods used to perform IVPC of a collaborative and intelligent automation system. Section 2 briefly explains the use-case and goes through the components showing the overall complexity of the system. A quick overview of traditional VC is extended with newer concepts of VC in Section 3. Section 4 touches upon humans as part of the loop in VC and discusses different methods of including humans in VC. Section 5 goes through the components of the control system and Section 6 shows implementation of some of those components. The paper is concluded in Section 7 with a short overview of the paper and the achieved results.

2 The collaborative and intelligent automation system use-case

It is challenging to develop collaborative and intelligent automation systems [7] since they involve smart algorithms, advanced sensors, human operators and collaborative robots. Because of this, it is almost impossible to develop all aspects of the system without IVPC.

In this paper, this is highlighted in a collaborative robot assembly station located in a Volvo Trucks engine manufacturing facility in Skövde, Sweden (shown in Figure 2).

This industrial use-case demonstrates the design of a workstation where both humans and robots work in a collaborative and co-active fashion. Particularly in this use-case, a collaborative robot and a human operator perform

Figure 2: The collaborative robot assembly station.

assembly operations on a diesel truck engine.

In the use-case station, an Automated Guided Vehicle (AGV) that carries a diesel truck engine and an autonomous mobile platform (MiR100) that carries the kitted material to be mounted on the engine, enter the collaborative robot assembly station. An engine ladder frame, three oil filters and several oil transport pipes are to be mounted on the engine in this station.

Before the collaborative execution of these operations can begin, an authorized operator has to be verified with a RFID reader. After verification, the operator is greeted by the station and instructions for tasks that are to be performed are shown on a screen. If no operator is verified, some operations can still be executed independently by the robot, however, violation of safety zones around the robot trigger a safeguard stop.

In the case of collaborative assembly, a dedicated camera system keeps track of the human assuring safe coexistence. During positioning of the AGV and the MiR100 in the assembly station, a Universal Robots (UR10) robot attaches to a special end-effector needed for manipulation of the ladder frame. Since the ladder frame is quite heavy, transporting it from the kitted MiR100 to the engine is done together by the robot and the human.

After placing the ladder frame on the engine, the operator informs the control system with a button press on a smart watch after which the UR10 leaves the current end-effector and attaches itself to a smart tool used for tightening bolts. During this tool change and as it is instructed on the screen,

the operator is placing twelve pairs of bolts needed for assembling the ladder frame on the engine and indicates completion of this task through the smart watch. Now, the control system knows that bolts are in place and the UR10 starts the tightening operation with the smart nutrunner.

Since tightening of these bolts takes some time, the operator can mount three oil filters during that time. If the robot finishes the tightening operation first, it leaves the smart tool in a floating position above the engine and waits for the operator.

The operator uses the smart watch again to let the system know that the oil filters are in place. This event makes the robot attach to a third end-effector and start performing the oil filter tightening operations. During the same time, the operator attaches two oil transport pipes on the engine, and uses the same smart tool that the robot has left floating to tighten plates that hold the pipes to the engine.

After executing these operations, the AGV with the assembled engine and the empty MiR100 leave the collaborative robot assembly station. A video showing the described use-case in action can be found here:
`https://youtu.be/YLZzBfY7pbA`

3 Traditional virtual commissioning

Simulation of production systems is a well adopted practice in modern industry. Industry leading software tools like Process Simulate and Delmia include support for VC, which exposes the simulation model to a real control system. Using these tools for designing control systems and performing VC has become a standard in industry.

Over the years, the VC community specified several commissioning configurations [13], [8] that resulted in terms like hardware-in-the-loop, reality-in-the-loop and constructive commissioning. As shown in Table 1, these specifications are defined by combinations of components being real or virtual.

Testing and integrating the physical production system with the real control system has been traditionally referred to as physical commissioning. However, in order to reduce the amount of on site man-hours during physical commissioning [14], the real control system is coupled with a simulation model of the production system creating a hardware-in-the-loop setup. This configuration is commonly known as VC [8].

Plant	Controller	Commissioning type
Real	Real	Real (physical) commissioning
Real	Virtual	Reality-in-the-loop commissioning
Virtual	Real	Hardware-in-the-loop commissioning
Virtual	Virtual	Constructive commissioning

Table 1: Traditional commissioning classification

Performing the inverse of VC can be beneficial in situations when debugging the control system is needed. In this case, the physical production system is controlled by a simulated controller in a setup known as reality-in-the-loop.

When designing a new control system, the natural way is to start with offline programming where all components are simulated. This configuration is also known as constructive commissioning [8].

Performing VC can reduce testing and integration time, as well as help detect undesired behavior before physical commissioning. However, it is usually the case that creating simulation models requires extensive modelling effort. Because of the cost associated with this effort, it is crucial that the created models provide as much additional value as possible.

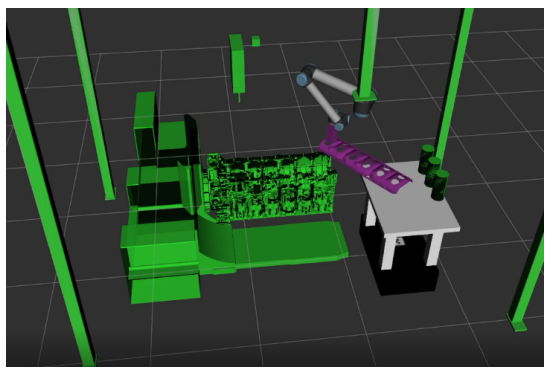


Figure 3: Early simulation work is beneficial for control system development.

This value can be increased by embedding VC into the engineering workflow. Instead of using VC as the last step before physical commissioning,

IVPC enables simulation supported production preparation and automation engineering by simultaneous development of the control system and the virtual plant [11]. Figure 3 shows an early stage simulation that serves as continuous support for development of the control system for the collaborative and intelligent automation system use-case.

The traditional commissioning classification does not really apply in complex systems that rely on intelligent control. Containing algorithms beyond standard executing sequences as described in Section 5, the control system is the component that needs commissioning.

Since the start of it's use, VC focused on highly automated systems where human interaction and interference has been limited. However, automation is today introduced in traditionally operator intensive areas such as assembly stations. This puts new requirements on VC.

4 Human-in-the-loop commissioning

Considering the role of humans in VC, especially in collaborative systems, human-in-the-loop commissioning has only recently become a topic of conversation [15].

In this chapter, three classes of the term human-in-the-loop are proposed, i.e. real, virtual and immersed. Always considering the use of the real controller, Table 2 shows the proposed human-in-the-loop (HITL) commissioning configurations.

Virtual-human-in-the-loop commissioning considers inclusion of a simulated human mannequin in the virtual plant environment. This can be beneficial for ergonomics verification [16], [17] as well as for safety testing using simulated safety equipment.

Plant	Human	Commissioning type
Real	Real	Real (physical) commissioning
Virtual	Virtual	Virtual-HITL commissioning
Virtual	Immersed	Immersed-HITL commissioning

Table 2: Human-in-the-loop commissioning classification

Means to detect simulated mannequins in a virtual plant involve simulated camera systems, light curtains, laser proximity sensors and other, where disturbances can also be allowed in sensor simulations. Moreover, behavior generation algorithms can ensure that the simulated mannequin covers large sets of different realistic kinematic behaviors [18].

Various HMIs can be emulated during IVPC in order to verify plant behavior during human interaction.

Humans can also be included in the commissioning loop by "immersion", which represents a real-time mapping from a real to a virtual human. Immersed-human-in-the-loop commissioning utilizes technologies like motion-tracking to "project" actual human kinematic behavior into the virtual plant environment. As it is shown in Fig. 4, [19], an Intelligently Moving Mannequin (IPS IMMA) [16], [17] maps actual human behavior that is inferred from a human tracking system into the virtual environment [6].

Interactable emulated and real HMIs can include buttons, screens, smart watches, eye movement and hand gesture tracking cameras, safety sensors, voice recognition and others to enable the immersed human to interface the system.

An immersion method that is worth mentioning is virtual reality (VR) [20] which can be combined with motion tracking to provide a "true immersion" experience. Additionally, platforms exist that support creation of industrial VR workspaces suitable for immersion of several humans [21].

A large amount of data can be collected during immersed commissioning which can later be used to drive behavior generation algorithms [22].

5 The control in collaborative and intelligent automation systems

From the challenges and requirements described so far, it is evident that there is a need for a control system that is able to step out of bounds of traditional automation methods and that can be commissioned using IVPC. In this paper, Sequence Planner (SP) [23] has been used as a tool for modeling, analyzing and control of large scale intelligent automation systems. SP includes supporting algorithms for a variety of use cases related to modeling, synthesizing control logic [24], formal verification, optimization, automated planning and visualizing complex operation sequences in different projections.

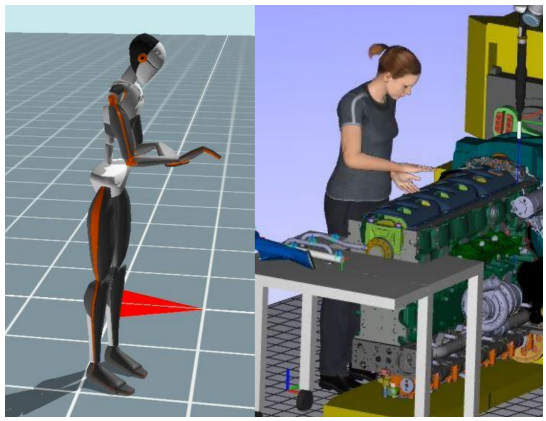


Figure 4: Motion capture immersion with IPS IMMA where motions can be translated from the same safety camera system feed shown in Fig. 1.

The core modeling aspects in SP are resources, abilities and operations. A resource is defined by a set of state variables that represents the current state of the resource. Some variables can be directly measured using sensors, others are used to represent the commands and some variables are used to estimate state that is not measured in the real system.

The resources have a set of abilities that they are able to perform. These abilities execute based on the current state of the resource and they update the command and estimated state variables. Abilities control everything in the system and are also used in the deliberation and execution loop of SP that is based on automated planning [25], which generates a plan deliberating what abilities to execute to reach a specific goal. These goals are defined by a number of high-level planning operations that continuously refine the current goal of the execution.

Most models in SP are based on transitions and variables [26], which are defined using extended finite automata (EFA) [27]. EFAs are finite automata extended with variables that are used in guards and actions associated with transitions. A transition in an EFA is enabled if and only if its corresponding guard formula (predicate) is true. After the transition is taken, a set of variables is updated by action functions. A transition table containing the behavior of an EFA models how an ability modifies the state of a resource in

the system, Table III.

predicate name	predicate (guard)	control actions	effects
enabled	$\neg run^c \wedge \neg run^m$	$run^c = T$	-
starting	$run^c \wedge \neg run^m$	-	$run^m = T$
executing	$run^c \wedge run^m \wedge \neg tqr^m$	-	$tqr^m = T \vee fail^m = T$
finished	$run^c \wedge run^m \wedge tqr^m$	$run^c = F$	-
failed	$run^c \wedge run^m \wedge fail^m$	$run^c = F$	-
resetting	$\neg run^c \wedge run^m$	-	$run^m = F$, $tqr^m = F$, $fail^m = F$

Table 3: The run nutrunner forward ability.

5.1 Resource

To exemplify abilities, let us consider the engine ladder frame tightening operation using the nutrunner described in Section 2. The nutrunner used in the industrial scenario described in Section 2 is quite complex. However, in order to exemplify the control of this system, a simplified version of the nutrunner is used from this point. The simplified nutrunner resource state variables can be defined as:

$$r_{nr} = \{run^c, run^m, tqr^m, fail^m\} \tag{B.1}$$

These Boolean variables make up the current state of the nutrunner, where run^c represents the command from SP to run the nutrunner forward and run^m is the signal from the nutrunner to SP showing if the toll is running forward or not.

After receiving the command to start running forward, the nutrunner engages tightening which will eventually result in reaching a specified torque tqr^m or failing $fail^m$.

5.2 Ability

The ability *run nutrunner forward* of the *nutrunner* resource models the task of tightening by starting to run the motors forward until a pre-programmed torque (tqr^m) has been reached or the tightening operation has failed $fail^m$. Table III shows the transitions of the *run nutrunner forward* ability, where each line makes up one possible transition of the ability.

For the purpose of this example, several lines are filtered out from ROS2 messages used for communication between SP and the nutrunner. As shown in the following listing, a *command* message is sent out from SP as a state based command to the nutrunner while a *state* message provides SP with the current state of the tool.

```
# /nutrunner/command
bool run_c              # run_tool_forward

# /nutrunner/state
bool run_c              # got_run_tool_forward
bool run_m              # tool_running_forward
bool tqr_m              # programmed_torque_reached
bool fail_m             # tool_failed
```

This listing shows only the measured and the command variables. Since the desired result of running the *run nutrunner forward* ability is to tighten a pair of bolts and there are no sensors for keeping track of bolt states, an *estimated* state variable $\hat{b} \in$ *(empty, placed, tightened)* can be introduced in order to keep track of the states of the bolts and enable the on-line planner to generate plans to tighten all bolts.

In order to maintain the simplicity of the example, the tightening of only one pair of bolts is considered. Now we can avoid the estimated bolt state and move on with the example.

The *enabled* predicate of the *run nutrunner forward* ability is declared as:

$$\neg run^c \wedge \neg run^m \tag{B.2}$$

However, this does not mean that the ability will start executing immediately after the guard is fulfilled by updating the variables with the corresponding control actions. This is due to the fact that the on-line planner deliberates if the control action should be executed or not.

5.3 Effect

To use the abilities in formal planning algorithms, measured variables must be updated so that the planner knows what to expect from the real system. In order to emulate an ability without its actual device or simulation, one or more starting and executing effects per ability are defined.

Effects model how measured state variables behave during execution of an ability. For example, issuing the command to start the nutrunner, the command variable run^c is set to true. Before the nutrunner actually responds that the tool has started to run forward, the *starting* predicate is now true. The effect mapped to this predicate models that the nutrunner will eventually be running forward.

After the tool has responded that it is running forward, the *executing* predicate is true. Now the effects model that bolts will eventually be either tightened or that the tightening operation will fail. These effects are extensively used during IVPC since they are used to emulate device behavior. This is explained in the following section.

6 Model-based ROS2 component generation

Based on abilities modeling the behavior of resources, SP's component generator module generates a set of ROS2 nodes during compile-time. Five nodes per resource are generated: *interfacer*, *emulator*, *simulator*, *driver* and *test*. Moreover, message types are also generated based on the model in SP as well as all other necessary ROS2 package components.

The *interfacer* node serves as the standardized interface node between SP and the nodes of the resource. *Emulator* nodes are completely auto-generated based on the abilities defined in SP and are used during IVPC.

Simulator and *driver* nodes implement the actual simulation and device control. *Test* nodes are generated based on the SP model and they implement automatic testing of *simulator* and *driver* nodes based on the SP model.

6.1 Emulator

The internal state of an emulation does not have to reflect the internal state of the target which it is emulating, it only has to mimic the observable behavior to match an existing target. Considering this, the internal structure of ROS2 *emulator* nodes is quite simple.

In order to emulate real device behavior during IVPC, effects happen outside of SP in dedicated ROS2 *emulator* nodes. These nodes emulate devices by executing effects specified in the SP model. It has to be noted that at a certain point in time, only one of the *emulator*, *simulator* or *driver* nodes per resource

is active.

In order to exemplify the *emulator* nodes, let us continue with the nutrunner example. After generating the nodes based on the model in SP, the *emulator* node now mimics the observable behavior of the modeled resource based on the effects specified in the model. This means that the node listens to commands from SP and publishes its state like a *simulator* or a *driver* node.

The *emulator* node contains the model of the resource, evaluating all predicates based on the messages received from SP. If some predicates are evaluated to be true, their corresponding effects are appended to a list of effects to be executed. After a command message has been received from SP and all predicates have been evaluated, the effects from the list of effects are executed.

This updates the *measured* variables which are then published from the node in the state message. This way, the model in SP can be continuously tested and improved using *emulator* nodes since the complete behavior of the resource based on the model in SP is achieved.

If the model changes during the development process, the *emulator* nodes are re-generated, capturing the change from the SP model. This shows that model-based code generation no only supports IVPC but is an essential part of it.

Moreover, since humans are resources in SP which have abilities that they can perform, human effects represent the expected human behavior after receiving an instruction. These human effects are contained in auto-generated human *emulator* nodes which emulate the human behavior based on the effects in the model.

6.2 Simulation

Contrasting to emulation, simulation involves modeling the underlying state of the target. Since collaborative and intelligent automation systems are composed of many different components, there is no single simulation environment that can support modeling of all aspects of the system.

ROS2 is a valid middleware to bridge different simulation environments with other ROS-based as well as non ROS-based components. This is supported by DDS being the robust communication layer in ROS2. There is a number of available implementations of DDS, and since DDS is standardized, ROS2 users can choose an implementation from a vendor that suits their needs. This is done through a ROS Middleware Interface (RMW), which also exposes Quality

of Service (QoS) policies.

QoS policies allow ROS2 users to adjust data transfer in order to meet the desired communication requirements. This feature is crucial, especially in distributed and heterogeneous systems, since setups are usually unique. Some of these QoS policies include setting message history and reliability parameters. These features are beneficial in a real system as well as when performing VC, since robust communication between simulation environments and other components can be challenging to achieve.

These simulation environments are interfaced with *simulator* nodes that are partially generated based on the model in SP. Initial SP interfacing templates are generated, however, interfacing details to the actual simulation environments have to be manually specified. This can be done via an import to the node since that way the common parts of the node can be re-generated if the model changes, without influencing the added interfacing part.

The virtual plant is implemented using ROS2, virtual machines (VMs) and Docker containers (DCs). VMs and DCs are used to segment the virtual model and mimic the distributed nature of the system. Since ROS2 does not rely on a rosmaster [28], physical segmentation of the virtual plant model is easily achieved in order to distribute the computational load between machines.

A software used during IVPC of the described setup that successfully implements the potential of the virtual world is Industrial Path Solutions (IPS) [5]. IPS is a mathematics based software tool for automatic verification of assembly feasibility, design of flexible components, motion planning and optimization of multi-robot stations and simulation of key surface treatment processes [5].

Figure 5 shows a virtual scene in IPS generated from the point cloud of the real assembly station. IPS is used as a tool to automatically generate collision free paths of the UR10 in the station on-line as well as off-line. In both cases, ROS2 serves as a middleware for IPS to communicate with the UR10, either with the real hardware or the simulator.

Moreover, having model based ROS2 component generation makes it straightforward to include a virtual or immersed human via the *simulator* or *driver* nodes to achieve human-in-the-loop IVPC.

Figure 5: The collaborative and intelligent automation system use-case in IPS.

6.3 Targeted IVPC and VM contained simulations

To extend the usability of code generation, multiple launch files are generated for various types of testing and commissioning purposes.

The first step in the engineering workflow would be testing the model with emulator nodes, iterating the model until the desired behavior is reached. Afterwards, a simulation or real hardware can be interfaced for one of the resources, while the others remain emulated. This way, real, simulated and emulated components can be combined in a mix-and-match fashion to achieve targeted model property testing.

This testing is supported using the *test* node that emulates SP by publishing model-based generated test commands, trying to break the behavior of the *simulator* node. After a failed test, the model is improved and the tests run again until they don't fail anymore.

Moreover, simulation environments and their accompanying *simulator* nodes can be contained in VM's and DC's to mimic the distributed nature of a real industrial setup.

7 Conclusion

This paper discusses IVPC of an engine assembly station where intelligent control algorithms are commissioned in a ROS2-based setup. To support IVPC, auto-generated ROS2 components are used to continuously test and improve the model in SP.

Moreover, due to ROS2's robust underlying communication layer DDS, it is seamless to mix real, simulated and emulated components during IVPC. As essential components of the intelligent control system, abilities and effects are briefly explained utilizing the description of the use-case provided in Section 2.

The concept of human-in-the-loop commissioning is discussed and a classification proposed where two different means of including humans in VC are isolated. A supporting literature review provides sufficient background to recognize the difference between virtual and immersed human-in-the-loop commissioning and a need to propose the mentioned classification. ROS2-based human-in-the-loop IVPC using auto-generated SP model-based emulator and simulator nodes and its benefits are discussed.

References

[1] P. Hoffmann, R. Schumann, T. Maksoud, and G. Premier, "Virtual commissioning of manufacturing systems - a review and new approaches for simplification," Jun. 2010, pp. 175–181, ISBN: 9780956494405.

[2] H. ElMaraghy, T. AlGeddawy, A. Azab, and W. ElMaraghy, "Change in manufacturing – research and industrial challenges," in *Enabling Manufacturing Competitiveness and Economic Sustainability*, H. A. ElMaraghy, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 2–9, ISBN: 978-3-642-23860-4.

[3] *Volvo GTO Vision*, `https://www.engineering.com/PLMERP/ArticleID/18868/Vision-and-Practice-at-Volvo-Group-GTO-Industry-40-and-PLM-in-Global-Truck-Manufacturing.aspx`, [Online; accessed 14-Apr-2019].

[4] R. Bloss, "Collaborative robots are rapidly providing major improvements in productivity, safety, programing ease, portability and cost

while addressing many new applications," *Industrial Robot: the international journal of robotics research and application*, vol. 43, no. 5, pp. 463–468, 2016.

[5] *IPS*, `http://www.fcc.chalmers.se/software/ips/`, [Online; accessed 14-Apr-2019].

[6] *Kinect ROS2*, `https://github.com/EresDavid/ROS-Kinect`, [Online; accessed 22-Apr-2019].

[7] A. Hanna, P.-L. Götvall, M. Ekström, and K. Bengtsson, "Requirements for designing and controlling autonomous collaborative robots system-an industrial case," *Advances in Transdisciplinary Engineering*, pp. 139–144, 2018.

[8] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[9] S. T. Mortensen and O. Madsen, "A virtual commissioning learning platform," *Procedia Manufacturing*, vol. 23, pp. 93–98, 2018, "Advanced Engineering Education and Training for Manufacturing Innovation"8th CIRP Sponsored Conference on Learning Factories (CLF 2018), ISSN: 2351-9789.

[10] M. Oppelt and L. Urbas, "Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering," *Proceedings, IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2564–2570, Feb. 2015.

[11] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: Supporting formal methods during automation systems development," *IFAC-Papers On Line*, vol. 49, no. 12, pp. 1939–1944, 2016, 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016, ISSN: 2405-8963.

[12] D. Lu, *The 2018 ROS Metrics Report*, `https://discourse.ros.org/t/the-2018-ros-metrics-report/6216/2`, [Online; accessed 25-Feb-2019], 2018.

[13] F. Auinger, M. Vorderwinkler, and G. Buchtela, "Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop"," in *WSC'99. 1999 Winter Simulation Conference Proceedings. 'Simulation - A Bridge to the Future' (Cat. No.99CH37038)*, vol. 1, 1999, 798–805 vol.1.

[14] N. Shahim and C. Møller, "Economic justification of virtual commissioning in automation industry," in *2016 Winter Simulation Conference (WSC)*, 2016, pp. 2430–2441.

[15] M. Metzner, J. Bönig, A. Blank, and J. Franke, ""human-in-the-loop"-virtual commissioning of human-robot collaboration systems," Apr. 2018.

[16] L. Hanson, D. Högberg, J. Carlson, *et al.*, "Imma – intelligently moving manikins in automotive applications," May 2014.

[17] D. Högberg, P. Castro, P. Mårdberg, *et al.*, "Dhm based test procedure concept for proactive ergonomics assessments in the vehicle interior design process: Volume v: Human simulation and virtual environments, work with computing systems (wwcs), process control," in Jan. 2019, pp. 314–323, ISBN: 978-3-319-96076-0.

[18] C. Esteves, G. Arechavaleta, J. Pettre, and J.-P. Laumond, "Animation planning for virtual mannequins cooperation," *ACM Transactions on Graphics - TOG*, Jan. 2004.

[19] P. Castro, D. Högberg, H. Ramsen, J. Bjursten, and L. Hanson, "Virtual simulation of human-robot collaboration workstations: Volume v: Human simulation and virtual environments, work with computing systems (wwcs), process control," in Jan. 2019, pp. 250–261, ISBN: 978-3-319-96076-0.

[20] M. Dahl, A. Albo, J. Eriksson, J. Pettersson, and P. Falkman, "Virtual reality commissioning in production systems preparation," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2017, pp. 1–7.

[21] P. Galambos, Á. Csapó, P. Zentay, *et al.*, "Design, programming and orchestration of heterogeneous manufacturing systems through vr-powered remote collaboration," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 68–77, 2015, Special Issue on Knowledge Driven Robotics and Manufacturing, ISSN: 0736-5845.

[22] M. Manns, S. Mengel, and M. Mauer, "Experimental effort of data driven human motion simulation in automotive assembly," *Procedia CIRP*, vol. 44, pp. 114–119, 2016, 6th CIRP Conference on Assembly Technologies and Systems (CATS), ISSN: 2212-8271.

[23] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Sequence planner: Supporting integrated virtual preparation and commissioning," *IFAC-Papers On Line*, vol. 50, no. 1, pp. 5818–5823, 2017, 20th IFAC World Congress, ISSN: 2405-8963.

[24] P. Ramadge and W. Wonham, "Wonham, w.m.: The control of discrete event systems. proc. ieee 77(1), 81-98," *Proceedings of the IEEE*, vol. 77, pp. 81–98, Feb. 1989.

[25] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*, 1st. New York, NY, USA: Cambridge University Press, 2016, ISBN: 1107037271, 9781107037274.

[26] M. Dahl, E. Erős, A. Hanna, K. Bengtsson, and P. Falkman, *Sequence planner - automated planning and control for ros2-based collaborative and intelligent automation systems*, `https://arxiv.org/abs/1903.05850`, 2019.

[27] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *2007 46th IEEE Conference on Decision and Control*, Dec. 2007, pp. 3387–3392.

[28] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*, 1st. O'Reilly Media, Inc., 2015, ISBN: 1449323898.

PAPER C

Evaluation of high level methods for efficient planning as satisfiability

Endre Erős, Martin Dahl, Petter Falkman and Kristofer Bengtsson

Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) vol. 26, 2021.

DOI: 10.1109/ETFA45728.2021.9613254

The layout has been revised.

Abstract

Fast planning algorithms play a key role in intelligent automation systems where control sequences are constantly calculated. In order to determine which algorithms increase planning performance, we evaluate and compare several high level planning methods on a set of standard benchmarks. We focus on planning as satisfiability as the leading approach for solving difficult planning problems.

1 Introduction

In order to create truly flexible automation systems, they need to include planning algorithms that can compute schedules and sequences of operations automatically. Taking correctness of planning algorithms for granted, efficiency is the key trait such algorithms must have in order to be applicable in the industry. Hence, this paper investigates and evaluates performances for some commonly used techniques in automated planning such as invariants, skipping steps and subgoaling.

Planning is a deliberative decision making process which yields sequences of operations that drive state change towards a goal. Planning as satisfiability, where planning problems are encoded as logical formulas and handled by SAT solvers [1] was introduced by Kautz and Selman in 1992 [2]. In a SAT based approach, a solver determines whether there exists a satisfiable assignment for a given Boolean formula that encodes the plan. It turned out to be quite a valuable contribution since SAT based planning excels in solving hard combinatorial planning problems with a high number of variables [3].

Over the last two decades, many methods have emerged with the aim to improve the efficiency of SAT solving [4] and SAT based planning [5]. Additionally, coupling a SAT solver with theory solvers, for example theories such as linear arithmetic, bit vectors, or arrays, SMT based planning techniques [6] can encode and tackle real-world scenarios and complex application domains [7].

In this paper, we investigate and compare several relevant methods for planning as satisfiability. We do not however study the detailed tuning of these solvers, but instead focus on what we call *high level* methods to improve

the planning performance, which for example include the structure of the model and how to call the solvers. To the extent of our knowledge, there are no other studies that perform such an evaluation and comparison of high level planning methods. Thus, we hope that our effort will be useful, particularly for those who aim to implement new satisfiability based planners.

The paper is organized in the following way. The next section presents, evaluates, combines and compares planning methods in an evolutionary way, using a number of standard planning benchmarks. Some drawbacks and special combinations are discussed in Section 3. Section 4 presents the combined results and the paper is concluded in Section 5.

2 Planning methods

In planning as satisfiability, a planning problem is encoded into a logical formula F_j, where j represents the number of plan steps. This formula is then tested for satisfiability by a solver, which either returns UNSAT for a failed planning attempt or it returns SAT and provides a satisfying assignment which is parsed to yield a plan. If the result from the solver is UNSAT, the planning problem is encoded into a logical formula of length $j+1$ and tested again by the solver. This goes on sequentially until the solver returns SAT with a satisfying assignment, or until a maximum limit on the plan length is breached. More formally:

Definition 1: *A planning problem Ψ with a plan length j can be encoded by the logical formula F_j:*

$$F_j = I(t_0) \wedge (\bigwedge_{k=0}^{j-1} \bigvee_{n=0}^{m} (T_n(t_k, t_{k+1}))) \wedge G(t_j) \tag{C.1}$$

where I are clauses that encode the initial state at time-step t_0, T_n are clauses that encode the m transitions of the model for all successive time-steps t_k and t_{k+1}, and G are clauses that encode the goal state at the horizon t_j.

We are going to explore how modifications to this planning approach affect the efficiency of planning, and in some cases ease modeling. In order to do that, we implement a set of algorithms which make use of different planning methods. Each subsection explores some methods and compares their performances on a set of classical planning benchmarks.

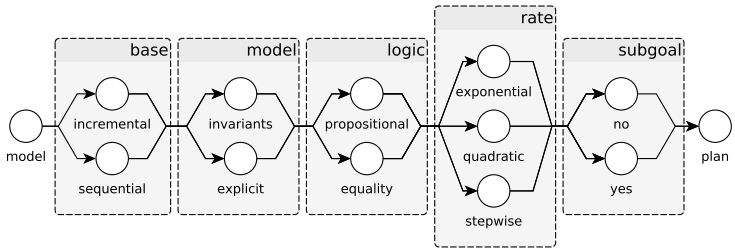


Figure 1: Planning methods. There is a total of 48 method combinations, however we do not investigate all combinations in this paper. Starting from the left side, we evolve our algorithms with methods shown in this figure.

The benchmarks that are chosen to test the methods are: *gripper* [8], *blocksworld* [9], *rovers* [10], *barman* [11] and *childsnack* [12]. These benchmarks are chosen to represent different strengths and weaknesses of the following planning methods. An overview of evaluated methods is shown in Figure 1.

2.1 Incremental vs. sequential base

As mentioned earlier, a satisfiability based planning algorithm sequentially increases the plan length until a plan is found or until a limit is breached. The sequential algorithm (Algorithm 1) takes a planning problem and returns a result that either holds a plan or is empty, which represents that no solution was found.

An integer variable *step* keeps track of the step in the plan that the algorithm is currently at. At line 2 of Algorithm 1, the main loop checks whether the limit on the plan length is breached. It is important to limit the plan length so that the algorithm can terminate in case a solution can't be found, or where it takes a long time to calculate it.

A context *ctx* is created at line 3 to which the algorithm asserts a number of constraints, as shown between lines 4 and 9. The solver now checks the consistency of all assertions in the context *ctx* and if a satisfiable assignment exists, it is parsed into a plan and returned by the algorithm at lines 13 and 14. Otherwise, if no satisfiable assignment exists, the *step* variable is incremented by 1 and the whole procedure repeats while the statement $step \leq s_{max}$ holds.

Algorithm 2: ***Sequential***

```
   Input: (i, g, M, s_max)
   Output: planning_result
 1 let step := 0;
 2 while step ≤ s_max do
 3 |   let ctx := new_context;
 4 |   add constraint (ctx, i, 0);
 5 |   add constraint (ctx, g, step);
 6 |   let m_disj := disjunction for trans in M;
 7 |   for s in (0 to step) do
 8 |   |   add constraint (ctx, m_disj, s);
 9 |   end
10 |   if check(ctx) == UNSAT then
11 |   |   step += 1;
12 |   else
13 |   |   let planning_result = parse(ctx.get_model);
14 |   |   return planning_result;
15 |   end
16 end
17 return new_empty_planning_result;
```

The downside of Algorithm 1 is that for every iteration where an assignment is not found, a new context has to be created. Gocht and Balyo showed in 2017 [13] that it is possible to achieve a significant speed-up by using an incremental SAT-solver. Instead of throwing away the context with all the accumulated data from previous results, the same context is used and constraints are just added on top.

Algorithm 3: ***Incremental***

```
   Input: (i, g, M, s_max)
   Output: planning_result
 1 let step := 0;
 2 let ctx := new_context;
 3 add constraint (ctx, i, step);
 4 let bp := new_backtracking_point;
 5 add constraint (ctx, g, step);
 6 while step ≤ s_max do
 7 |   step += 1;
 8 |   if check(ctx) == UNSAT then
 9 |   |   ctx.backtrack_to_level_bp;
10 |   |   let m_disj := disjunction for trans in M;
11 |   |   add constraint (ctx, m_disj, step);
12 |   |   let bp := new_backtracking_point;
13 |   |   add constraint (ctx, g, step);
14 |   else
15 |   |   let planning_result = parse(ctx.get_model);
16 |   |   return planning_result;
17 |   end
18 end
19 return new_empty_planning_result;
```

The incremental algorithm (Algorithm 2) utilizes an incremental solver which makes it possible to add a *backtracking point* in each step so that the solver can choose which part of the context to save, and thus learn from previous attempts. In summary, some advantages of an incremental base solver are:

1. *Learnt clauses are kept*
2. *Heuristic data is gathered*
3. *Overhead from asserting the same clauses is reduced*

At line 3 of Algorithm 2, the initial constraints for *step-0* are asserted into the context. Before asserting the goal, the algorithm creates a backtracking point so that if no satisfiable assignment was found, only the goal can be removed from the context, leaving the initial constraints.

At line 8, the algorithm checks the consistency of assignments in the context. If a satisfiable assignment is found, it is parsed and returned. Otherwise, the algorithm backtracks to the latest point, removing the assertions added after it from the context.

At line 11, the transitions are added, after which a new point is made so that the algorithm can backtrack to it if the assignments are not consistent with the goal assignment. This goes on until a plan is found or until a limit on the plan length is breached. Figure 2 compares planning efficiency between these two algorithms, where one utilizes an incremental solver and another one does not.

2.2 Invariants vs. explicit model

In the previous subsection, it was concluded that using the incremental base algorithm can substantially improve planning efficiency. From now on, we will only use the incremental algorithm as our base.

Invariants can be considered both as a modeling aid as well as a performance increasing method in automated planning. Invariants are specifications that must hold in every step of the calculated plan. More formally:

Definition 2: *An invariant is a logical clause that must hold for all reachable states of Ψ, and can be encoded in the the logical formula F_j:*

$$F_j = I(t_0) \wedge (\bigwedge_{k=0}^{j-1} \bigvee_{n=0}^{m} T_n(t_k, t_{k+1})) \wedge (\bigwedge_{k=0}^{j} N(t_k)) \wedge G(t_j)$$

where I are clauses that encode the initial state at time-step t_0, T_n are clauses that encode the m transitions of the model for all successive time-steps t_k and t_{k+1}, N are clauses that encode invariants for all time-steps t_k, and G are clauses that encode the goal state at the horizon t_j.

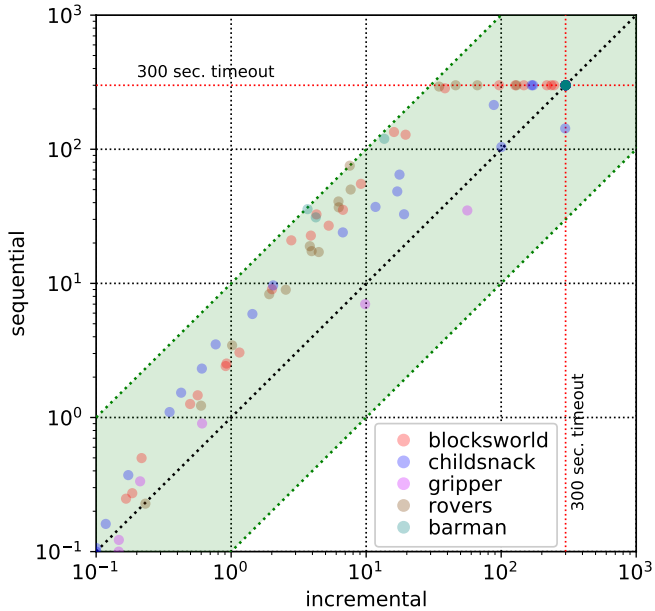


Figure 2: This scatter plot shows that the algorithm that utilizes incremental solving is usually much faster in solving problems than the non-incremental algorithm because learned clauses are kept in the context. 73 (incremental) vs 61 (sequential) out of 200 instances were solved, where 97% are solved faster using the incremental base algorithm.

In essence, invariants prune states from the state space. They can be added to the model to forbid some undesired behavior, or to enforce tighter constraints and thus derive a more compact state space representation. In each case, the state space gets reduced and thus planning is more efficient.

For an existing problem, invariants can sometimes be synthesized to speed up planning [14]. However, invariants can often be a convenient tool to use for modelling different problems, but can in some cases be quite hard to use. Nevertheless, in order to illustrate how modelling using invariants can improve planning efficiency, let's look at a well known PDDL benchmark example, Blocksworld. The Blocksworld example is one of the most famous planning domains in artificial intelligence.

Imagine a set of blocks sitting on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved [9].

Invariants that should hold at any given time for the Blocksworld example are added to this problem:

1. b1 can't be on b2 if b2 is on b1
2. if holding any block, the gripper can't be empty
3. at most one block can be held
4. a block can't be on several different blocks
5. if a block is on the table, it is not on a block
6. if b1 is on b2, b2 is not clear

Now we have a much safer and smaller state space representation, and as an effect, planning is much faster. Figure 3 shows the difference between planning times with and without making use of planning invariants for a number of standard planning benchmarks.

2.3 Equality vs. propositional logic

In the previous section, it was concluded that invariants have a beneficial effect for both the modeling and planning efficiency. From now on, we consider problems that are reinforced with additional invariants, if that is indeed possible.

Beside invariants, there are other things that can aid modeling. By using different *first-order* logic theories to increase expressiveness, some problems can be modeled in a much more convenient way compared to using pure propositional logic. In this paper, we only investigate equality logic since it is appropriate for modelling most classical planning problems.

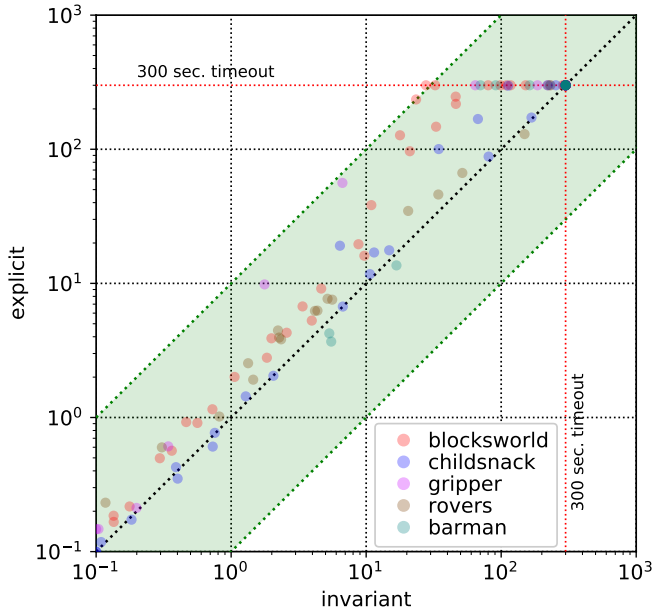


Figure 3: On this scatter plot, it is shown that modelling the problem using invariants reduces the state space and thus speeds up planning , especially in the harder cases. 90 (invariants) vs 73 (explicit) out of 200 instances solved, where 97% are solved faster using invariants.

Using a first-order theory with increased expressiveness to ease modeling can potentially lead to decreased decidability. Let's investigate the expressiveness and decidability aspects of both propositional and equality logic theories, in order to determine the beneficial effects on modelling and planning performances. A more comprehensive text on the problem of expressiveness vs. decidability, as well as decision procedures in general can be found in [15].

Definition 3: *The following grammar defines the syntax of formulas in propositional logic:*

$$formula : formula \wedge formula \mid \neg formula \mid atom$$
$$atom : Boolean - identifier \mid true \mid false$$

Example

Let's continue with the Blocksworld example and look at an action that is modeled in pure propositional logic:

$$
\begin{aligned}
&(: \mathit{action}\ \mathit{pick\_up}\\
&\qquad : \mathit{parameters}\ (?x\ -\ \mathit{block})\\
&\qquad : \mathit{precondition}\\
&\qquad\ (\mathit{and}\ (\mathit{clear}\ ?x)\\
&\qquad\qquad (\mathit{ontable}\ ?x)\\
&\qquad\qquad (\mathit{handempty}))\\
&\qquad : \mathit{effect}\\
&\qquad\ (\mathit{and}\ (\mathit{not}\ (\mathit{ontable}\ ?x))\\
&\qquad\qquad (\mathit{not}\ (\mathit{clear}\ ?x))\\
&\qquad\qquad (\mathit{not}\ (\mathit{handempty}))\\
&\qquad\qquad (\mathit{holding}\ ?x))
\end{aligned}
$$

This action describes picking up a block from the table. In order to pick the block x up, the block has to be *ontable* and *clear* for picking up or placing another block on top of it. The hand has to be empty, as indicated by the *handempty* variable. After the block has been picked up, it is no longer *ontable*, nor is it *clear*. The hand is no longer empty, it is now *holding* the block x.

As it can be seen, to model this action for every block x, there has to be as many Boolean variables for *holding*, *ontable* and *clear* as there are defined blocks.

Let us now study the equality logic and the Blocksworld example using finite domain variables. If we restrict ourselves to using only finite domain variables, equality logic can be thought of as propositional logic where the atoms are equalities between variables or between variables and constants. A more formal definition of equality logic follows:

Definition 4: *The following grammar defines the syntax of formulas in*

equality logic:

$$
\begin{aligned}
&formula : formula \wedge formula \mid \neg formula \mid atom \\
&atom : term \ = \ term \\
&term : identifier \mid constant
\end{aligned}
$$

where the identifiers are variables defined over a single finite domain of values. These values can be a subset of the set of Integers or Reals. Just the same, the variables can be of an enumeration type and define their own finite domain. Constants are elements from the same domain as identifiers.

Example

The same *pick_up* action modeled using equality logic could look something along the lines of:

$$
\begin{aligned}
&(: action\ pick\_up \\
&\qquad : parameters\ (?x\ -\ block) \\
&\qquad : precondition \\
&\qquad\ (and\ (clear\ ?x) \\
&\qquad\qquad (=\ (on\ ?x)\ \ ''table'')) \\
&\qquad : effect \\
&\qquad (and\ (not\ (clear\ ?x)) \\
&\qquad\qquad (=\ (on\ ?x)\ \ ''hand''))
\end{aligned}
$$

Now, instead of using the Boolean variables *holding* and *ontable* for every block as well as the *handempty* variable, we can use just one enumeration type variable *on* with a finite domain:

$$v_{on}^{D} = blocks \cup \{''table'',\ ''hand''\}$$

Both propositional and equality logic are NP-complete [16], [17], which means that they can model the same decision problems with not more than a polynomial difference in variables [15]. Certain problems are more conveniently modeled in equality logic compared to propositional logic, and for some other problems the opposite is true.

As for efficiency, the high level structure in the input equality logic formula can potentially be used to make the decision procedure work faster. This information may be lost if the problem is modelled directly in propositional logic [15].

Figure 4 compares planning efficiency between problems modeled in propositional logic and equality logic on 2 planning benchmarks.

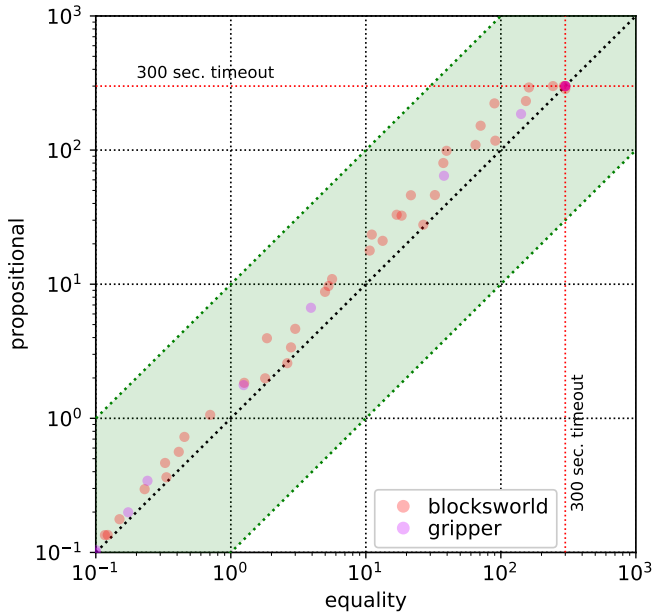


Figure 4: This scatter plot shows that modelling problems using equality logic can sometimes increase planning efficiency. In some other cases, there is no high level structure that can be exploited by modelling the problem in equality logic, thus there is no big difference in performance. Thus, the appropriateness of using equality over propositional logic depends on the problem itself.

2.4 Skipping steps

As it was mentioned before, the planning algorithms increment the plan length by one when an assignment is not found, after which the encoding for that

length is tested for satisfiability. This is a good method when the yielded plan should be of minimal length. However, calculating the shortest length plan can sometimes be very slow, as Rintanen showed in [18] (shown in Figure 5).

The evaluation cost of an unsatisfiable formula can be much higher than the evaluation cost of a satisfiable one, even if the latter is not of shortest length. Especially, when considering the sum of evaluation costs for all unsatisfiable formulas, it can be seen that the planner can spend a lot of time trying to find a satisfiable assignment. This is because the cost of evaluating the unsatisfiable formulas usually increases exponentially as the plan length increases [18].

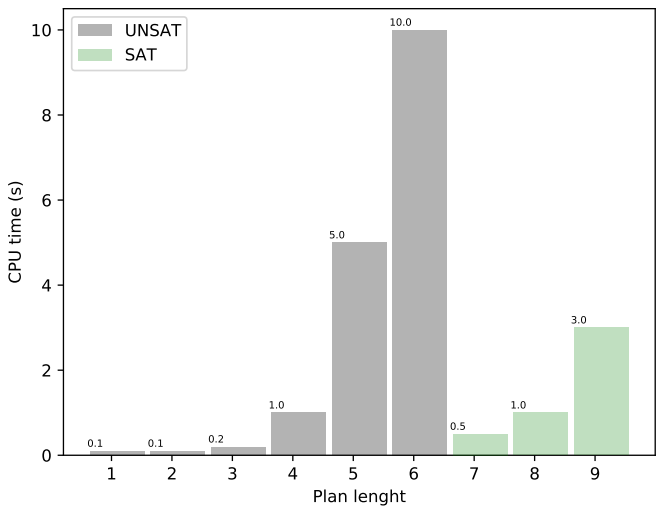


Figure 5: Evaluation cost of the unsatisfiable formulas for plan lengths 1 to 6 and the satisfiable formulas for plan length 7 and higher, from [18]

.

When finding the first satisfiable solution which yields the plan with the minimal length is not a strict requirement, an improvement in the planning time can sometimes be achieved. Rather than increasing the plan length by one after an assignment is not found, this improvement is realized by incrementing the plan length by a larger value. This allows us to skip some hard unsatisfiable instances which take a long time to evaluate.

As mentioned, skipping steps does not guarantee that the shortest plan will

be yielded. For example, if we choose to evaluate only the encodings for the plan lengths 1, 2, 4 and 8 from Figure 5, we will skip the hard unsatisfiable instances 5 and 6, but also the first satisfiable instance 7. The plan length will be of length 8 instead of 7, but the benefit will be the decrease in planning time, 2.2s instead of 16.9s.

Incrementing the plan length after an assignment is not found can, for example, evolve in some constant or exponential rate. One could choose to solve for constantly increasing plan lengths $2i$ or $5i$ for integers $i \geq 1$, quadratic i^2, or for exponentialy increasing lengths 2^i for integers $i \geq 1$.

Figure 6 compares planning efficiency between planning algorithms that utilize different rates of skipping steps when using an incremental solver. Figure 7 compares the results when using a sequential solver.

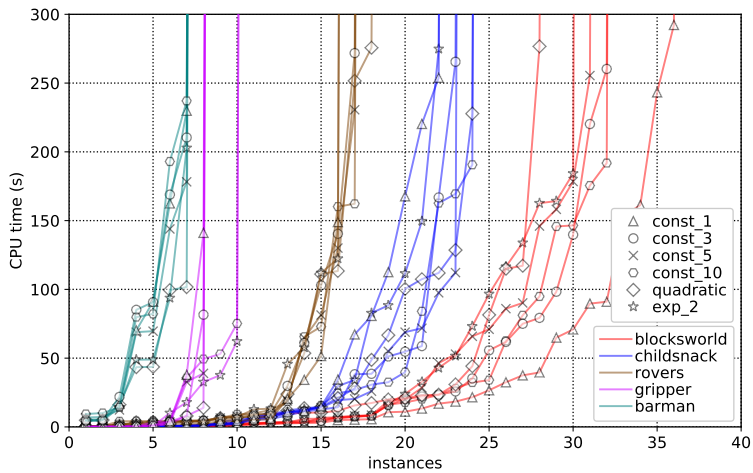


Figure 6: This cactus plot shows the planning times of algorithms that skip planning steps in a certain way. The algorithms are essentially the same, it is only the step skipping rate that differs in them. Since skipping of steps is implemented on top of the incremental base algorithm, the solver has not learnt anything from the skipped iterations. In some cases, and as visible for the Blocksworld instance, skipping steps actually negates the good effects of using the incremental base solver. This will be further discussed later.

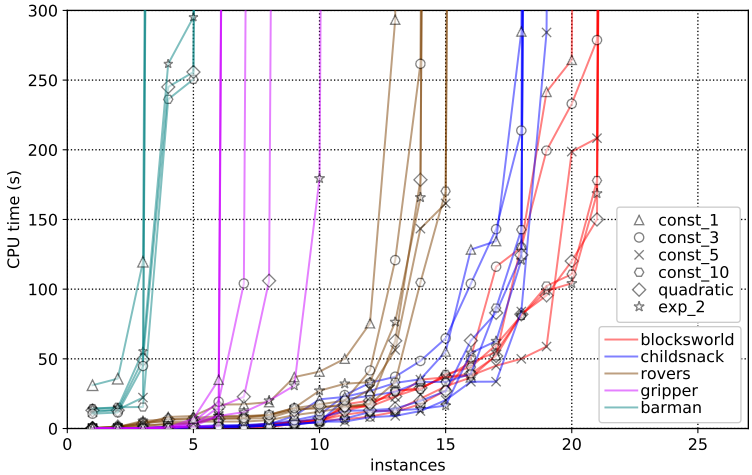


Figure 7: The beneficial effect of skipping steps if much more visible if the base solver is non-incremental. This means that there is no drawback of skipping steps because the solver has not learnt anything from the skipped clauses.

2.5 Subgoaling

Skipping steps seems to help for some problems, and for others it does not. As we will discuss later, this is because we use the incremental base algorithm. Thus, we will continue to increment the rate stepwise.

When a goal is defined as a conjunction of several predicates, such predicates can be looked at as *subgoals*. Instead of asking the planner to reach the monolithic goal in one planning task, multiple planning tasks can be instantiated to plan for each subgoal. Having a simpler goal shortens the plan length and thus the planning time. As mentioned before, the cost of evaluating unsatisfiable formulas increases exponentially, thus it is usually faster to search for a large number of shorter plans than the other way around.

Algorithm 3 shows how subgoaling utilizes the incremental algorithm to solve for each conjunct of the monolithic goal. If that was indeed possible, the algorithm concatenates the results to yield the plan for the monolithic problem in line 18.

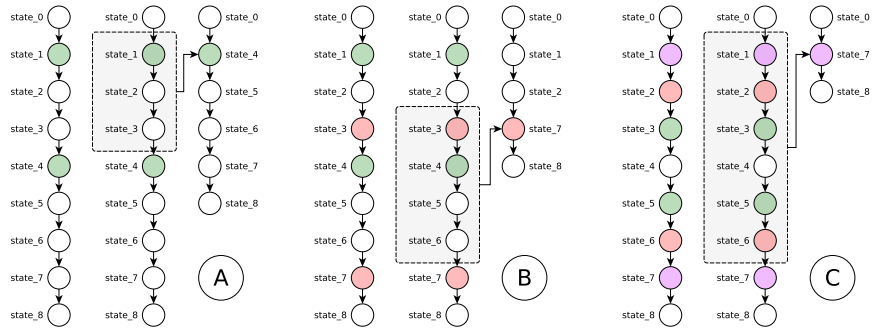


Figure 8: Shorten plan scenarios. In scenario A, there is only one section of the plan that leads back to a same state (states 1 - 4), so the part is removed. In scenario 2, there are two overlapping loops, so in order to be more efficient, the algorithm removes the bigger loop. There might be loops still in the plan after removal so in the next iteration, the algorithm checks the plan for more loops. In scenario C, there is some nesting of loops, so the algorithm finds the biggest loop and removes both in one go.

Subgoal ordering matters

If the subgoal order is not correct, finding a plan can sometimes be slower or even impossible. This depends quite much on the nature of the problem itself as planning problems often exhibit symmetry properties that could be exploited to speed up their solving.

For example, the two highly symmetrical problems Gripper and Childsnack benefit the most from subgoaling. There is no important subgoal order that has to be enforced in order to get a correct plan. On the other hand, the Blocksworld problem relies heavily on a correct order of subgoals.

2.6 Shortening the plan length

When planning with methods that skip steps or use subgoaling, yielded plans often have more steps than necessary. Sometimes, it is possible to remove chunks from a plan. An efficient way to do this is to detect loops in the plan and remove sections of it that lead back to an already visited state.

Usually, the requirement to visit the same state more than once is tracked

Algorithm 4: ***Subgoal***

Input: *(i, G, M, s_{max})*
Output: *planning result*

```
1  let subresults := new_empty_list;
2  let first_sub := Incremental(i, G[0], M, s_max);
3  let n := 0;
4  subresults.push(first_sub);
5  Plan(first_sub, subresults, n, i, G, M, s_max);
6  function Plan(r, subresults, n, i, G, M, s_max) begin
7  if n < G.len() − 1 then
8  |  n = n + 1;
9  |  let new_i := if r.trace.len() == 0 then
10 |  |  i;
11 |  else
12 |  |  result.trace.tail();
13 |  end
14 |  let sub := Incremental(new_i, G[n], M, s_max);
15 |  subresults.push(sub);
16 |  Plan(sub, subresults, n, i, G, M, s_max);
17 else
18 |  return Concatenate(subresults);
19 end
```

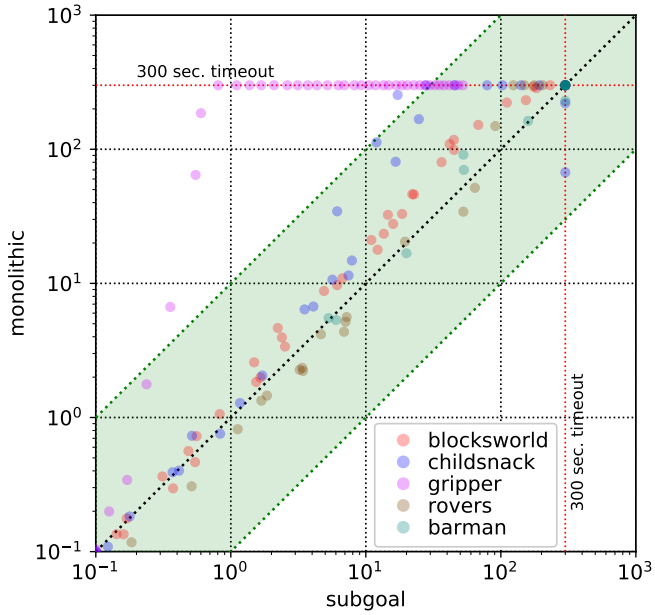


Figure 9: On this scatter plot, it is shown that subgoaling will usually decrease planning time, especially in highly symmetrical problems.

by an additional variable, hence the states in the trace have the information about the complete state. If all variables are considered in a state, there should not be two of the same states in a plan. Having this in mind, we can remove parts of the plan to yield a valid plan of shorter length. Algorithm 4 shows how redundant sections of a plan are removed.

At line 5 of Algorithm 4, the *Find* function searches the plan for duplicate states and saves them, as well as their location in the plan, to a list. If duplicate states do exist, they cover a certain section of a plan that has to be removed. It can be that more than two duplicate states exist, as well as more than one pair of duplicates.

If that is the case, the section they cover can overlap in some way, as shown at B and C parts of Figure 8. Hence, the function *GetBiggest* finds the biggest section covered by two duplicates and the function *Remove* removes it from the plan. The plan is now shortened. However, it can be that it still contains

some duplicate states. Because of this, the *Shorten* algorithm performs all the mentioned steps recursively until it exhausts all duplicates from a plan.

Algorithm 5: ***Shorten***

Input: *plan of length n*
Output: *plan of length m, m ≤ n*

```
1  Shorten(plan);
2  function Shorten(plan) begin
3  let duplicates := new empty list;
4  for frame in plan do
5  |  duplicates.push(Find(frame.state, plan));
6  end
7  if not duplicates is empty then
8  |  let biggest := GetBiggest(duplicates, plan));
9  |  let new_trace := RemoveLoop(biggest, plan));
10 |  Shorten(new_plan);
11 else
12 |  return plan;
13 end
```

2.7 Benchmarks

We ran the benchmarks on an Optiplex 9020 desktop PC with 8GB of RAM and an Intel Core i7-4790 CPU clocked at 3.60GHz. All algorithms used in this study were implemented using Z3's [19] quantifier free finite domain (QF_FD) theory, which supports propositional logic, bit-vector theories, pseudo-Boolean constraints, and enumeration data types. Figure 10 shows how different methods have improved the planning algorithm throughout the paper.

3 Discussion

Throughout this paper, we have investigated the effects of incremental solving, invariants, equality logic, skipping steps and subgoaling. We have done this in an evolutionary way, where a method from each subsection has improved the

existing algorithm from the previous subsection. However, what about other method combinations and their performances?

What we learned from this study is that modelling with invariants always helps, both in easing the modeling itself and decreasing planning time. If we take that as a starting point of this discussion, we have to look at methods that fit well together with invariants. For example, incremental solving lets us keep some clauses in the context before we move on to the next step. Since invariants hold in each step of the plan, the clauses that encode them are never removed from the context.

Continuing on this, if we use an incremental base solver and skip some steps, the solver will not have the chance to learn new clauses from the skipped steps. However, what if we turn things around and use the sequential base together with skipping steps? Figure 7 shows the results of this combination. It can be seen that these results better match Rintatnen's [18] chart on Figure 5, thus it is more appropriate to combine non-incremental solving with skipping steps. Using non-incremental solving with skipping steps is probably beneficial if the computation is distributed among several cores. However, single-core planning benefits more using incremental solving, as seen on Figures 6 and 7.

Beside methods investigated in this paper, there is a multitude of other methods that weren't considered in this study. Probably one of the biggest contributors to speed in planning as satisfiability is parallel planning and it is presented in detail in [20]. Moreover, there is compositional planning as investigated in [21], and hierarchical planning [22].

Big performance improvements in planning as satisfiability can be unlocked when methods are altered on the low, SAT solving level. Some of these methods improve decision heuristics [23], restart heuristics [24] and data structures [25]. We refer to these methods as low level methods since they affect the performance of the solver itself. As such, they are of major interest in the field of planning as satisfiability, however they are out of the scope for this paper.

4 Conclusion

We implemented and evaluated several high level planning methods using Z3 and showed how much each method contributes to an increase in planning performance. As discussed, this is by no means a complete study that cov-

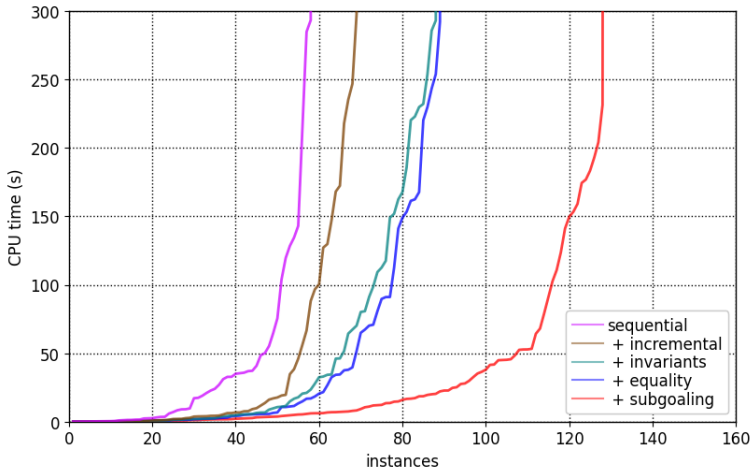


Figure 10: Algorithm evolution throughout the paper.

ers all planning methods, however we feel that it is a useful reference and performance overview of some high level methods.

That being said, we see that there is great opportunity in studying automated planning, satisfiability, and method combinations that improve planing performance. Our next step is to use what we learned in this study in actual industrial implementations, as well as to investigate other high level methods that we mentioned in the discussions. Moreover, an investigative survey of low level planning methods, i.e. methods that operate on the SAT solving level is a planned step in our future work.

Acknowledgment

This research is supported by Chalmers University of Technology and VINNOVA under the projects Unification (contract nr. 2017-02245) and Unicorn (contract nr. 2017-03055).

References

[1] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518, ISBN: 978-3-540-24605-3.

[2] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the 10th European Conference on Artificial Intelligence*, ser. ECAI '92, Vienna, Austria: John Wiley & Sons, Inc., 1992, pp. 359–363, ISBN: 0471936081.

[3] J. Rintanen, "Search methods for classical and temporal planning," *Tutorials of the 21th European Conference on Artificial Intelligence (ECAI 2014)*, vol. 21, 2014.

[4] S. Alouneh, S. Abed, M. H. A. Shayeji, and R. Mesleh, "A comprehensive study and analysis on sat-solvers: Advances, usages and achievements," *Artificial Intelligence Review*, pp. 1–27, 2018.

[5] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012, ISSN: 0004-3702.

[6] J. E. Arxer, "Smt techniques for planning problems," 2018.

[7] A. Bit-Monnot, F. Leofante, L. Pulina, and A. Tacchella, "Smt-based planning for robots in smart factories," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali, Eds., Cham: Springer International Publishing, 2019, pp. 674–686, ISBN: 978-3-030-22999-3.

[8] D. M. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, no. 2, p. 35, Jun. 2000.

[9] F. Bacchus, "Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems," *AI Magazine*, vol. 22, no. 3, p. 47, Sep. 2001.

[10] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis.," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 1–59, Dec. 2003.

[11] A. Coles, A. Coles, A. Olaya, *et al.*, "A survey of the seventh international planning competition," *Ai Magazine*, vol. 33, pp. 83–88, Mar. 2012.

[12] M. Vallati, L. Chrpa, M. Grześ, *et al.*, "The 2014 international planning competition: Progress and trends," *AI Magazine*, vol. 36, no. 3, pp. 90–98, Sep. 2015.

[13] S. Gocht and T. Balyo, "Accelerating sat based planning with incremental sat solving," in *ICAPS*, 2017.

[14] J. Rintanen, "An iterative algorithm for synthesizing invariants," in *AAAI/IAAI*, 2000.

[15] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*, 2nd ed. Springer Publishing Company, Incorporated, 2016, ISBN: 978-3-662-50497-0.

[16] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71, Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158, ISBN: 9781450374644.

[17] D. Kozen, "Positive first-order logic is np-complete," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 327–332, 1981.

[18] J. Rintanen, "Evaluation strategies for planning as satisfiability," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser.
ECAI'04, Valencia, Spain: IOS Press, 2004, 682–686
, ISBN: 9781586034528.

[19] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3.

[20] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as satisfiability: Parallel plans and algorithms for plan search," *Artificial Intelligence*, vol. 170, no. 12, pp. 1031–1080, 2006, ISSN: 0004-3702.

[21] E. Erős, M. Dahl, P. Falkman, and K. Bengtsson, "Towards compositional automated planning," Sep. 2020, pp. 416–423.

[22] D. Schreiber, D. Pellier, H. Fiorino, and T. Balyo, "Efficient sat encodings for hierarchical planning," Jan. 2019, pp. 531–538.

[23] J. Rintanen, "Heuristics for planning with sat," vol. 6308, Sep. 2010, pp. 414–428, ISBN: 978-3-642-15395-2.

[24] J. Huang, "The effect of restarts on the efficiency of clause learning," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07, Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2318–2323.

[25] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001
, pp. 530–535.

PAPER D

Towards compositional automated planning

Endre Erős, Martin Dahl, Petter Falkman and Kristofer Bengtsson

Published in conference proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) vol. 25, pp. 416–423, 2020.

DOI: 10.1109/ETFA46521.2020.9212040

The layout has been revised.

Abstract

The development of efficient propositional satisfiability problem solving algorithms (SAT solvers) in the past two decades has made automated planning using SAT-solvers an established AI planning approach. Modern SAT solvers can accommodate a wide variety of planning problems with a large number of variables. However, fast computing of reasonably long plans proves challenging for planning as satisfiability. In order to address this challenge, we present a compositional approach based on abstraction refinement that iteratively generates, solves and composes partial solutions from a parameterized planning problem. We show that this approach decomposes the monolithic planning problem into smaller problems and thus significantly speeds up plan calculation, at least for a class of tested planning problems.

1 Introduction

Increased industrial competitiveness requires fundamental changes in automation. As companies introduce collaborative, intelligent and flexible systems into production to meet the variability, quality and punctuality needs of the modern customer, production requirements make it evident that traditional automation solutions can't solve all challenges [1].

Machines nowadays operate in *complex* and *dynamic* environments, *quickly* producing a wide *variety* of *quality* products for *demanding* customers. As *these* requirements continue to increase, it becomes impossible to remain scalable and sustainable using traditional automation solutions [2].

Instead, modern automation solutions should utilize efficient planning algorithms that compute schedules and sequences of operations automatically. Planning is a deliberative decision making process which yields sequences of operations that drive state change towards a goal.

Automated planning is already implemented in some modern solutions, however the idea of planning isn't new since algorithms that compute plans based on explicit state-space searches exist since the late '50s [3], and symbolic methods based on BDDs since the late '70s [4].

A more recent method that has established itself as an important planning approach is SAT-based planning. Planning problems are encoded as satisfiability problems and the results are calculated by SAT solvers.

Even though SAT-based planning was first proposed by Kautz and Selman already in 1992 [5], the interest of planning researchers in SAT-based planning methods was limited up until recently. One of the main reasons behind this was the performance advantage of explicit state-space search over solving early SAT encodings of planning problems [6]. However, modern planners based on satisfiability match, and often outperform, planners based on other search paradigms [7].

Explicit state-space search and symbolic methods based on BDDs are known for their performance in solving problems with a small number of state variables, however SAT-based methods excel in solving hard combinatorial planning problems with a relatively high numbers of state variables [8]. Another advantage of planners based on SAT is that algorithms used to compute plans are almost completely general purpose SAT solving algorithms, which means that every improvement in the solver directly improves planning.

However, a known issue with SAT-based automated planning is that plan calculation seems to slow down significantly as the plan length increases [6]. In an effort to avoid this limitation while utilizing the strengths of SAT-based planning, we present a compositional algorithm that divides a planning problem into a number of simpler problems that are faster to solve. We do this with a combination of abstraction refinement using activation parameters and step-wise problem generation, resolution and concatenation. We test the proposed algorithm on a number of examples and show that in a lot of cases a significant speed-up [9] is achieved.

In the following section, we present a version of an existing incremental planning algorithm that we use as the low-level solving engine in our approach. In Section 3, the high-level compositional algorithm is introduced. In the section after that, we test our approach on a number of examples and compare planning performances of the compositional algorithm against the algorithm of Section 2. We discuss certain benefits, drawbacks and possible future improvements of the compositional algorithm in section 5. There, we also mention relevant publications in the area of compositional planning and planning with abstraction refinement. Finally, we conclude the paper in Section 6.

2 Incremental Planning

We utilize a simple incremental planning algorithm based on [10] to solve individual problems generated by the compositional algorithm. The incremental algorithm tries to find a plan by testing the satisfiability of the planning problem's formulas for a sequentially increasing horizon length. It utilizes an incremental SAT-solver that makes it possible to add a new time point in each step so that the SAT-solver can learn from previous attempts.

The version presented in this paper also lets the algorithm create backtracking points that enables it to manipulate the content of the context between each step, i.e. to remove certain clauses that are added after a point by backtracking to that point. For example, if a planning attempt fails, this feature is used to remove goal clauses for the current step before adding new goal clauses for the next step. Before describing the incremental algorithm, it is necessary to define a few basic concepts. Let's do this with an example.

Example

A robotic manipulator transports products from a buffer to a fixture where the products are processed. In order to control the robot, we have to read its current *pose* as well as to send *pose* commands to it. To establish this two way communication, a *pose command* and a *pose measured* variable is used. Let's call these variables $pose_c$ and $pose_m$.

The real sensor-level measurement from the robot is discretized into three discrete values that can be assigned to the variable $pose_m$. The robot can be at the *buffer (b)*, at the *fixture (f)* or at *unknown (u)* if it is somewhere between the *buffer* and the *fixture*.

Definition 1: *A variable v_i is a named unit of data that can be assigned a value from its finite domain of values v_i^D.*

Example

In our robot scenario, the values *buffer*, *fixture*, and *unknown* constitute the finite *domain* of the variable $pose_m$. We don't want to command the robot to go to the *unknown* pose, so only the poses *buffer* and *fixture* make up the domain of the variable $pose_c$.

Definition 2: *A complete variable set V_c for a system is a set of all variables defined for that system. A partial variable set V_p is a non-empty subset*

of V_c.

Definition 3: *The complete state space V_c^D of a system is the Cartesian product of all value domains of variables in V^c defined for that system. A partial state space V_p^D is a non-empty subset of V_c^D.*

Example

With the currently defined variables in our robot system, the *complete variable set* and the *complete state space* of our system is:

$$V_c = \{pose_m, pose_c\}$$
$$V_c^D = \{\langle b, b\rangle, \langle f, b\rangle, \langle u, b\rangle, \langle b, f\rangle, \langle f, f\rangle, \langle u, f\rangle\}$$

Now that we have defined the states our system can be in, it is time to model how the system can move between those states. Before we can do that, we have to define *predicates*, which are the fundamental building blocks of our *models*.

Definition 4: *A predicate is a logical expression of one or more variables. A predicate evaluates to true if the assignments of its variables satisfy the logical expression, and to false otherwise.*

Example

Let's build a few simple predicates and name them so that we can reuse them later. Something we might reuse a lot is knowing whether the robot is at the buffer, fixture or somewhere between:

$$r\_at\_b := pose_m == b$$
$$r\_at\_f := pose_m == f$$
$$r\_at\_u := pose_m == u$$

These predicates can either be *true* or *false*, depending on the current value of the variable $pose_m$ during evaluation. We can also build more complex predicates and name them as we like. For example, another thing we might find useful during modeling is knowing whether the robot has been issued a command to move to the *buffer*. With the current *complete variable set* that we have defined for this system, we can say that the robot is moving towards

the buffer if:

$$r\_moving\_to\_b := \neg r\_at\_b \wedge pose_c == b \tag{D.1}$$

Before we define *transitions*, let's look at how a very simple plan would look like. As it was said before, a *plan* is a sequence of transitions driving state change towards a goal. In this example, the robot moves from the *buffer* to the *fixture*.

$$\begin{aligned}
&state_0 : pose_m == b \wedge pose_c == b \\
&trans_1 : start_move_robot_to_fixture \\
&state_1 : pose_m == b \wedge pose_c == f \\
&trans_2 : finish_move_robot_to_fixture \\
&state_2 : pose_m == f \wedge pose_c == f
\end{aligned}$$

This sequence has several steps with each step being one transition leading to a state. The chain of states is called a *trace* and since only one transition is allowed to be taken at a time, the states are *temporally* related. This means that we can say that $state_2$ is the *next* state after $state_1$.

Definition 5: *A transition t is a predicate:*

$$t = g \wedge e \tag{D.2}$$

where g is a guard predicate and e is an effect predicate. If the transition is to be taken, the guard predicate has to evaluate to true for the current step, while at the same time, the effect predicate has to evaluate to true for the next step.

Example

Let's look at the start_move_robot_to_fixture transition. This transition is modeled as:

$$\begin{aligned}
&start_move_robot_to_fixture = \\
&pose_{m_i} == b \wedge pose_{c_i} == b \wedge pose_{c_{i+1}} == f
\end{aligned}$$

where the guard predicate is marked with i for the current step and the effect predicate with $i+1$ for the next step. We finally have the necessary components to define a planning problem.

Algorithm 6: ***Incremental***

```
   Input: (i, g, T, s_max)
   Output: planning result
 1 let step := 0;
 2 let ctx := create context;
 3 add constraint (ctx, i, step);
 4 let bp := create backtracking point;
 5 add constraint (ctx, g, step);
 6 while step ≤ s_max do
 7 |   step += 1;
 8 |   if check(ctx) == UNSAT then
 9 |   |   backtrack to level bp;
10 |   |   let t_disj := disjunction for T;
11 |   |   add constraint (ctx, t_disj, step);
12 |   |   let bp := create backtracking point;
13 |   |   add constraint (ctx, g, step);
14 |   else
15 |   |   return planning result;
16 |   |   break;
17 |   end
18 end
19 return empty planning result;
```

Definition 6: *A transition system T for a given system is a collection of all transition predicates that model the behavior of that system.*

Definition 7: *A planning problem Ψ is a 4-tuple:*

$$\Psi = \langle i, g, T, s_{max} \rangle \tag{D.3}$$

where i and g are initial and goal predicates, T is the transition system, and s_{max} is a limit on the horizon length. Actually, Ψ is a 5-tuple containing S as well, where S is a collection of specifications modeled as LTL_f formulas encoded in SAT [11]. However, specifications are omitted in this paper for the sake of brevity, so we will refer to a planning problem as it is defined in (D.3).

The incremental algorithm takes a planning problem Ψ and either returns a complete result of the planning problem, or an empty result which represents that no solution was found.

An integer variable *step* keeps track of the step in the plan that the algorithm is currently at. At line 2 of Algorithm 1, a context *ctx* is created for the problem so that the solver can keep track of assertions.

As you can see at lines 3 and 5, the algorithm *asserts* the initial and goal constraints for *step-0* into the context. It also creates a backtracking point in line 4 before asserting the goal, so that it can be removed from the context if a solution is not found in the current step.

Inside a loop that ensures the horizon limit is not exceeded, the algorithm increments the *step* and *checks* if the current assertions in the context are consistent. If the assertions are SAT in the first step, that means that the variable assignments satisfy both the goal and the initial predicates.

Otherwise, the assignments in the goal predicate for *step-0* are not consistent with the assignments in the initial predicate, so the goal is for the current step is removed them from the context by backtracking to the previous point.

The solver checks the transitions in a step from the disjunction of all transitions in the model. If any transition in the disjunction satisfies the current assignment, *planning* goes on. Semantically, it can be said that the transition is *taken*.

Only one transition is allowed to be taken in each step from the disjunction of all transitions in the model. Practically, transitions are tracked in each step with Boolean-valued variables, so that by the time a plan is found, we know which transition was evaluated to *true* in which step. This is done by conjuncting the transition with a Boolean-valued variable, so if the transition is *taken* in a step, that variable has to be true in that step.

The algorithm doesn't know if the next goal assignment will be consistent with the assignments that are currently in the context, so it creates a new backtracking point before it assigns the goal for *step-1* into the context.

As you can see, this process is repeated while the horizon length sequentially increases. If a solution for the problem is found in a *step* that is less than the horizon length limit s_{max}, it is returned by the algorithm. Otherwise, the limit is breached and an empty result is returned.

It is important to limit the planning horizon so that the algorithm can terminate in case a solution can't be found, or where it takes a long time to calculate it.

3 Compositional Planning

As mentioned before, the main idea behind the compositional algorithm shown as Algorithm 2 is to break the planning problem into simpler problems that can be solved fast. To solve these individual simple planning problems, the incremental algorithm from the previous section is used.

Figure 1 serves as an example and a visual guide to explain how the compositional algorithm works and what is happening in different steps. To keep track of these steps, we refer to the alphabetic annotations on the right side of the same figure.

3.1 Organization

In order for the compositional algorithm to refine, generate and solve parts of the complete planning problem, we have to allow its abstraction and refinement. The planning problem (D.3) is *parameterized* so that the compositional algorithm can enable or disable certain *basic predicates* in order to generate abstracted input constraints to the incremental algorithm. In order to do this, a number of *partial variable sets* is defined during modeling.

Definition 8: *A basic predicate is a predicate of variables from only one partial variable set.*

Defining partial variable sets depends on some expert knowledge of the planning problem, hence it is a part of the modeling process. For example, we choose to group variables together into partial variable sets so that we can form basic predicates that contain none other but variables from the same set.

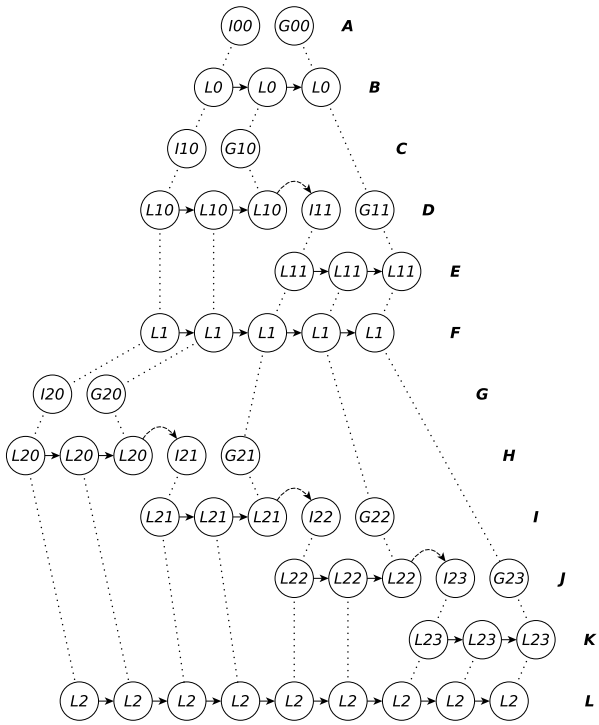


Figure 1: How the compositional algorithm works.

This allows us to compose more complex *parameterized predicates* and at the same time keep track of which partial variable sets play a role in them.

Example

Let's extend our example by adding more functionality to our system. For example, as a safety feature, let's add the option to control the robot's *status* which can enable or disable its movement. For this, we define two new variables, $stat_c$ and $stat_m$ with the same value domain $\{enabled, disabled\}$.

Moreover, let's equip the robot with a sensor in order to know whether it is holding a product or not. In fact, let's equip also the buffer and the fixture with the same type of sensors so that we can always track where the products

are. For this, we define three additional variables, $grip_m$, $buff_m$ and $fixt_m$ with the same value domain $\{empty, full\}$.

In this example, grouping the variables into partial variable sets comes naturally. Let's name these sets to make it easier to reuse them later:

$$pose = \{pose_c, pose_m\}$$
$$stat = \{stat_c, stat_m\}$$
$$prod = \{grip_m, buff_m, fixt_m\}$$

A good example of a *basic predicate* would be (D.1) since it only contains variables from the *pose* partial variable set. In order to know which variables are in a basic predicate, the name of the partial variable set they belong to will be present as a superscript in the names of basic predicates:

$$r\_moving\_to\_b^{pose} := \neg r\_at\_b \wedge pose_c == b$$

We have defined the building blocks that enable us to generate abstracted planning problems for the incremental algorithm.

3.2 Parameterization

In order to enable or disable certain *basic* predicates, we define *activation parameters* that enable or disable parts of *parameterized predicates.*

Definition 9: *An activation parameter is a Boolean-valued variable that is used to hide (false) or reveal (true) basic predicates in a parameterized predicate.*

Definition 10: *A parameterized predicate is a set of 2-tuples:*

$$\{\langle bp_1, a_1\rangle, \langle bp_2, a_2\rangle, ..., \langle bp_n, a_n\rangle\} \tag{D.4}$$

where $bp_1, bp_2, ..., bp_n$ are basic predicates and $a_1, a_2, ..., a_n$ are their respective activation parameters.

One activation parameter is usually defined for each partial variable set. Hence, an alternative notation will be used for parameterized predicates throughout the rest of the paper:

$$\{bp_1^{a_1}, bp_2^{a_2}, ..., bp_n^{a_n}\} \tag{D.5}$$

Definition 11: *A parameterized transition t_P is a transition whose guard and effect are parameterized predicates.*

Example

Let's build a parameterized transition that models how the robot takes products from the buffer.

$$\begin{aligned} &take\_product\_from\_buffer = \\ &\{b\_full_i^{prod}, r\_empty_i^{prod}, active_i^{stat}, activate_i^{stat}, \\ &\ r\_at\_b_i^{pose}, r\_go\_to\_b_i^{pose}, r\_full_{i+1}^{prod}\} \end{aligned}$$

In order for the robot to hold a product in the next step ($r\_full_{i+1}^{prod}$), several things have to be fulfilled in the current step. The buffer must hold a product ($b\_full_i^{prod}$) that the empty robot can take ($r\_empty_i^{prod}$). In order to do anything, the robot has to be active ($active_i^{stat}$) and in order to take a product from the buffer, it has to be at the buffer ($r\_at\_b_i^{pose}$).

Meanwhile, we don't want to cause changes in some variables in the next state. The basic predicate ($activate_i^{stat} := stat_{c_i} == enabled$) ensures that the measured variable $stat_m$ will hold its *enabled* value in the next step. The same goes for the $pose_m$ variable.

Definition 12: *A parameterized transition system T_P is a transition system whose transitions are parameterized transitions.*

Definition 13: *A parameterized planning problem Ψ_p is defined as:*

$$\Psi_P = \langle i_P, g_P, T_P, P, s_{max} \rangle \tag{D.6}$$

where i_P and g_P are initial and goal parameterized predicates, T_P is a parameterized transition system, P is a list of activation parameters and s_{max} is the limit on the plan length that is applied to every generated problem sent to the incremental algorithm.

Now that we have modeled a parameterized planning problem, we let activation parameters enable or disable basic predicates in parameterized predicates, generating abstracted input constraints for the incremental algorithm by conjuncting these enabled basic predicates.

3.3 Activation

The compositional algorithm takes a list of activation parameters P, activates the next parameter in the list, generates and solves problems. Hence, the order of the activation parameters in the list P decides the problem refinement order.

Example

Let's take the parameter list:

$$P = (prod, stat, pose)$$

and $take\_product\_from\_buffer$, the parameterized transition that we have built earlier. This transition contains seven basic predicates, where the first six are constituting the guard, and the last one the effect. While generating the real input predicate for the incremental algorithm as a conjunction of these basic predicates, the values of their respective activation parameters determine whether the basic predicate is included in the conjunction or not.

The activation parameter list P has three parameters that are initially disabled, hence the *Activate* procedure from Line 3 of Algorithm 2 activates the first parameter *prod*, enabling the basic predicates $b\_full_i^{prod}$, $r\_empty_i^{prod}$ and $r\_full_{i+1}^{prod}$ to take part in the generated conjunction that is the input transition for the incremental algorithm.

We can refer to this step-wise parameter activation as *refinement*. Each step of the compositional algorithm is called a *level*, and in each level, the problems that are generated and sent to the incremental algorithm are more refined, meaning that more variables play a role in the generated predicates of a problem. For instance, let's follow the complete refinement of the generated transition:

$$b\_full_i^{prod} \wedge r\_empty_i^{prod} \wedge r\_full_{i+1}^{prod}$$

In the next level, the algorithm activates the *stat* parameter, so the generated transition is more refined:

$$b\_full_i^{prod} \wedge r\_empty_i^{prod} \wedge$$
$$active_i^{stat} \wedge activate_i^{stat} \wedge r\_full_{i+1}^{prod}$$

Finally, in the last level, the algorithm activates the *pose* parameter and the generated transition is completely refined. The completely refined transition is a conjunction of all predicates from its parameterized counterpart.

3.4 Generation

As you can see from line 4 of Algorithm 2, the *Plan* function receives the planning result of the previous level, and for each step in the calculated plan

Algorithm 7: ***Compositional***

Input: $(i_P, g_P, T_P, P, s_{max})$
Output: *planning result*

```
1  let level := 0;
2  let (i, g, T, P) := Activate(i_P, g_P, T_P, P);
3  let r := Incremental(i, g, T, s_max);
4  Plan(r, i_P, g_P, T_P, P, s_max, level);
5  function Plan(r, i_P, g_P, T_P, P, s_max, level) begin
6  if not all parameters activated then
7      if r.plan_found then
8          let h := := new empty list;
9          let concat := 0;
10         let level_results := new empty list;
11         (i, g, m, P) := Activate(i_P, g_P, T_P, P);
12         for j in (0 to (r.trace.length - 1)) do
13             let g_l := r.trace(j + 1);
14             concat += 1;
15             if j == 0 then
16                 let r_P := Incremental(i, g_l, T, s_max);
17                 let h := r_P.trace.tail;
18                 level_results.push(r_P);
19             else if j == r.trace.length - 1 then
20                 let r_P := Incremental(h, g, T, s_max);
21                 level_results.push(r_P);
22             else
23                 let r_P := Incremental(h, g_l, T, s_max);
24                 let h := r_P.trace.tail;
25                 level_results.push(r_P);
26             end
27         end
28         let r_loopy := Concatenate(level_results);
29         let r_filt := Filter(r_loopy);
30         Plan(r_filt, i_P, g_P, T_P, P, s_max, level + 1);
31     else
32         return empty planning result;
33     end
34 else
35     return r;
36 end
```

it generates a new planning problem. This generation doesn't occur at once, since the following problem depends on the result of the previous problem in the same level. We could say that the following problem *inherits* information from the previous result.

Three cases are differentiated while generating problems depending on the place of the step in the solution of the previous level. We refer to these cases as the *first*, *central* and *last* problems of a level. A *first* case problem is generated from the first step in the plan of the previous level. For example, this can be seen at point C in Figure 1. Similarly, a *last* case problem comes from the last step in the plan of the previous level as it can be seen at point D. All other problems that are generated from steps between the first and the last step are *central*. This can be seen at points H and I in Figure 1.

Steps in the plan hold transitions that change the state towards a goal. Sample plans at certain levels of the compositional algorithm can be seen at lines B, F and L in Figure 1. Each transition in a plan has a *source* and *sink* state. The three different cases of problems in a level come from the way the *initial* and *goal* predicates of a problem are generated from these *source* and *sink* states of a transition in a step.

First case problem

When the algorithm is generating a *first* case planning problem, the initial predicate i of this problem is generated from the initial parameterized predicate i_P after the next parameter has been activated. In essence, it is a refined version of the initial predicate from the previous level. This is happening at line 16 of Algorithm 2.

The goal predicate of the *first* case planning problem is generated from the *sink* state of the transition in the first step, and it is a conjunction of assignments that make up that state. You should note that this new *goal* predicate doesn't contain variables that play a role in the initial predicate of the same problem. In order to concatenate results after they are calculated, the goal predicate of the *first* case planning problem is not provided with assignments for the variables that are being included in this level.

If we would to provide assignments for these variables in the goal predicate of the *first* case problem, there would probably be a discrepancy between assignments of adjacent plans of the level during concatenation. We would have to guess the assignments for those variables and in the end, we would

have to plan between plans to achieve a correctly concatenated solution. That is why we leave it to the solver to decide an assignment to the new goal predicate variables included in the next level that is consistent with the other assignments.

Last case problem

When the algorithm uses the last step in the plan of the previous level to generate a problem, that is a *last* case planning problem. This is happening at line 20 of Algorithm 2 and at points D and J in Figure 1. Since no further problems will follow in this level, the goal predicate g of this problem is generated from the goal parameterized predicate g_P after the next parameter has been activated. In essence, it is a refined version of the goal predicate from the previous level.

The initial predicate of the *last* case planning problem is generated from the *source* state of the transition in the last step, and it is a conjunction of assignments that make up that state. This time, we are providing the *initial* predicate with additional assignments by using an *inheritance* variable h to pass assignments between problems and solutions of a level, as you can see in lines 18 and 25 of Algorithm 2. At this point, after the solver has found satisfiable assignments for the goal state of the previous problem of the same level, the new assignments are *inherited* by the next planning problem to play a role the *initial* predicate, as you can see at points *D, H, I* and J in Figure 1.

Central case problem

When the algorithm generates a *central* case planning problem, the initial predicate is generated the same way as the initial predicate in the *last* case problem, and the goal predicate the same way as the goal predicate in the *first* case problem.

3.5 Resolution

To solve each generated problem of a level, we use the Algorithm 1 from the previous section. In lines 4, 17, 21 and 24 of Algorithm 2, this is indicated with *Incremental*. You can see in lines 16, 20 and 23 of Algorithm 2 how the algorithm distinguishes problem cases based on where the step of the plan of

the previous level is located. For example, *first* case problems are generated at points C and G and solved at points D and H in Figure 1. Similarly, *last* problems are generated at points D and J and solved at points E and K, and *central* problems are generated at points H and I and solved at points I and J in Figure 1. The important thing to notice is that in each level, problem generation and resolution happen iteratively one after the other until all solutions have been found for that level, or until one of the resolutions fail. This iterative generation, inheritance and resolution can best be seen between points G and K in Figure 1.

3.6 Concatenation

In each level, new planning problems are generated from the result of the previous level. As these problems are solved, we end up with a number of results that we have to concatenate in order to get the complete result of that level. This concatenation of plans represents the complete plan of the current level, and it is indicated with *Concatenate* in line 29 of Algorithm 2, which is happening at points F and L in Figure 1.

To keep track of where results should fit in the complete result of a level, we use a *concat* variable, as you can see in line 15 of Algorithm 2. After all problems of a level are solved, we concatenate the individual plans in the right order using the *concat* variables to get the plan of the level.

3.7 Filtering

In some cases, the result after concatenation might contain certain sections that are redundant. These sections lead to a duplicate of a state that was already reached earlier in the trace, so you can look at these sections as *loops*.

If loops appear in the concatenated result, the algorithm filters them out of the plan using the *Filter* procedure as you can see in line 30 of Algorithm 2. These *loops* appear sometimes after concatenation as a result of solving individual problems of a level while satisfying all specifications at that level. In planning, it is not of our interest to visit a logical state more than once, so if a loop appears in the concatenated trace, the algorithm removes it. After we filter out the loops, the plan is still consistent with all specifications at that level.

Once all parameters are activated, the problem is completely refined. After

the last check of the parameter list as seen in line 6 of Algorithm 2, if a plan is found in the previous level, it is returned by the algorithm in line 35.

4 Evaluation

Several planning problems are solved using both the compositional and the incremental algorithms. In this section, we are evaluating some of the results and comparing the length and quality of plans calculated by the incremental and the compositional algorithm.

4.1 Example 1

Let's test the algorithms on the robot example used in this paper. For brevity, the names of partial variable sets are shortened to one letter: *s* for *stat*, *i* for *prod* and *p* for *pose*.

For all the variations in the refinement order, the compositional algorithm calculates the same 18 step long plan as the incremental algorithm. The different benchmarks are derived from at least 10 runs for each test and the name of the test suggests the refinement order in the example. You can see from the benchmarks that the refinement order influences the plan calculation time, which is expected.

4.2 Example 2

Let's extend Example 1 with several additional complications. Now, we are able to control and measure the pose of the gripper, as well as to control and measure its status. Additionally, in order to avoid collision, the robot has to move through three *via* points. In this example, we have five partial variable sets: *s* for *stat*, *i* for *prod*, *p* for *pose*, *g* for *grip_pose* and *m* for *grip_stat*.

Provided is a short list of results that yielded the same 38 step long plan while testing this example. You can see from the benchmarks that as much as a 20x speed-up can be achieved with a good refinement order.

```
Benchmark:          Time (mean ± dev):
test_2_inc          13.461 s ± 0.500 s
test_2_comp_mgsip   1.026 s  ± 0.011 s
test_2_comp_gmpsi   908.4 ms ± 7.2 ms
```

```
test_2_comp_gmisp  671.3 ms ± 9.8 ms
```

5 Discussion

We found several publications that share similar opinions with us to be, to the best of our knowledge, state-of-the art results in areas of compositional planning and planning with abstraction refinement.

The authors of [12] use a counterexample guided abstraction refinement method to solve planning problem instances encoded in SAT. They obtain a relaxed instance by removing clauses from the model that is in conjunctive normal form and refine it later using counterexamples found during the search.

In [13], the authors distinguish between planning problems with and without symmetries. They decompose a planning problem with symmetries into a set of abstracted, isomorphic subproblems. After solving each abstraction, they concatenate the results together to yield a solution for the original problem.

This paper is our first contribution in a series that tries to tackle the problem of safe and non-blocking online planning, and as such, it only focuses on computing a plan faster. Safety and non-blocking properties are well known problems, however they will be addressed in future papers in order to limit the scope of this work.

What we show is that there are cases when the compositional algorithm clearly outperforms the well known incremental algorithm in terms of computation time. However, there are a few known shortcomings which we will mention now.

5.1 Refinement order matters

As you can see from the benchmarks in Examples 1 and 2, defining a good refinement order can influence the planning time quite much. Solving the same problem with a different refinement order gives quite different plans before filtering out the loops.

Usually, after filtering out the loops, the plans calculated after these two refinement orders are the same. However, it happens sometimes that the refinement order influences the final plan length as well. In these cases, the yielded plan is still correct, however it is not of the minimal length.

5.2 No optimality guarantee

If there are several valid plans of different lengths that can be calculated at a certain step, a shortest one will be yielded. However, this doesn't mean that this will yield the shortest plan of a level. In some cases, the compositional algorithm can provide a plan in a level that is short, however after refinement it would turn out that the solution after the last refinement is longer than the result the incremental algorithm would yield. This happens because of the breadth-first search nature of the incremental algorithm.

6 Conclusion

This paper presents a high-level compositional implementation that utilizes the an incremental SAT-based planning algorithm as its solving engine to perform automated planning. This is achieved by generating abstracted problems from the main parameterized planning problem, solving them using the incremental solver and concatenating the results in order to achieve a complete plan. We show that in a majority of cases, a significant speed-up is achieved.

Primarily, our plan for the future is to research the influence of the refinement order in this approach. We would like to develop a rule about defining a good refinement order, since solving a problem for all refinement order variations is not feasible, even for a small number of activation parameters.

Secondly, we will investigate safety and non-blocking properties of compositional planning and lastly, we would like to test the algorithm on standard planning benchmarks.

References

[1] A. Hanna, K. Bengtsson, M. Dahl, E. Erős, P. Götvall, and M. Ekström, "Industrial challenges when planning and preparing collaborative and intelligent automation systems for final assembly stations," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 400–406.

[2] A. Azizi, *Applications of Artificial Intelligence Techniques in Industry 4.0*, 1st. Springer Publishing Company, Incorporated, 2018, ISBN: 9789811326394.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X.

[4] S. Akers, "Binary decision diagrams," *Computers, IEEE Transactions on*, vol. C-27, pp. 509–516, Jul. 1978.

[5] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the 10th European Conference on Artificial Intelligence*, ser. ECAI '92, Vienna, Austria: John Wiley & Sons, Inc., 1992, pp. 359–363, ISBN: 0471936081.

[6] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012, ISSN: 0004-3702.

[7] ——, "Madagascar: Scalable planning with sat," *Proceedings of the 8th International Planning Competition (IPC-2014)*, vol. 21, 2014.

[8] ——, "Search methods for classical and temporal planning," *Tutorials of the 21th European Conference on Artificial Intelligence (ECAI 2014)*, vol. 21, 2014.

[9] ——, "Evaluation strategies for planning as satisfiability," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser. ECAI'04, Valencia, Spain: IOS Press, 2004, 682–686
, ISBN: 9781586034528.

[10] S. Gocht and T. Balyo, "Accelerating sat based planning with incremental sat solving," in *ICAPS*, 2017.

[11] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13, Beijing, China: AAAI Press, 2013, pp. 854–860, ISBN: 9781577356332.

[12] N. Froleyks, T. Balyo, and D. Schreiber, "Pasar - planning as satisfiability with abstraction refinement," in *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCs 2019), Napa, CA, July 16-17, 2019*, AAAI Press, Menlo Park, CA, 2019, pp. 70–78, ISBN: 978-1-57735-808-4.

[13] M. Abdulaziz, C. Gretton, and M. Norrish, "A verified compositional algorithm for ai planning," in *ITP*, 2019.