International Journal of Computer Integrated Manufacturing

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tcim20

Programming cobots by voice: a pragmatic, web-based approach

Tudor B. Ionescu & Sebastian Schlund

To cite this article: Tudor B. Ionescu & Sebastian Schlund (2022): Programming cobots by voice: a pragmatic, web-based approach, International Journal of Computer Integrated Manufacturing, DOI: 10.1080/0951192X.2022.2148754

To link to this article: https://doi.org/10.1080/0951192X.2022.2148754

Published online: 26 Nov 2022.

INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING
https://doi.org/10.1080/0951192X.2022.2148754

OPEN ACCESS

Programming cobots by voice: a pragmatic, web-based approach

Tudor B. Ionescu and Sebastian Schlund

Human-Machine Interaction, TU Wien, Vienna, Austria

ABSTRACT
This paper introduces a novel voice-based programming approach and software framework for collaborative robots (cobots) based on the Web Speech API, which is now supported by most modern browsers. The framework targets human programmable interfaces and human-machine interfaces, which can be used by people with little or no programming experience. The framework follows a meta-programming approach by enabling users to program cobots by voice in addition to using a mouse, tablet, or keyboard. Upon a voice instruction, the framework automates the manual tasks required to manipulate the vendor-provided interfaces. The main advantages of this approach are simplified, guided programming, which only requires the knowledge of 5–10 voice instructions; increased programming speed compared to the manual approach; and the possibility of sharing programs as videos. The approach is generalized to other kinds of robots and robot programming tools using so-called meta-controllers, which leverage the power of graphical user interface automation tools and techniques.

ARTICLE HISTORY
Received 17 July 2021
Accepted 13 November 2022

KEYWORDS
Voice-based programming; GUI automation; cobots; plugins; meta-programming; speech recognition

1. Introduction

Assembly automation has increasingly developed over the last decades also thanks to the introduction of collaborative robotic arms (short, cobots), which can work in close proximity to humans without requiring a safety fence. Cobots are currently used to automate repetitive manual tasks, such as loading/unloading of machines, pick & place operations, screwing tasks, or quality inspection. Cobots can help to accomplish some of these tasks more cost effectively than humans in productive industrial contexts. Nevertheless, the field of hybrid automation – i.e. where humans and machines work together in direct interaction has not been realized to the extent projected by the industrial automation community and cobot vendors.

Currently, a number of factors sum up to prevent the wide-scale adoption of cobots in Europe. First, current robot programming environments are usually considered counterintuitive because they build upon older expert programming models, such as function blocks, textual programming, and combinations of visual and textual programming modes. At the same time, newer robot programming environments, which draw upon design patterns used for smartphone user interfaces and other commonly used day-to-day technologies, provide opportunities for non-experts to engage with programming cobots more easily (Ionescu and Schlund 2019). Yet, while the barriers are lowered, these environments often lack the versatility of textual programming and are subjected to app-oriented business models, which foresee the acquisition of additional robot apps (or skills) at prohibitive costs. Second, while robot vendors strive for targeting new markets and users, in the industrial domain, robot programming is still regarded as being a task for experts. This leads to a gap concerning a learner's transition from multi-model, intuitive robot programming towards the more versatile, textual programming environments. Such a transition would, for example, allow assembly workers to program robots using increasingly complex programming models.

In response to this situation, this paper considers a scenario in which robot programming is certified as an assembly application instead. This would allow for workers to design applications in collaboration with assembly planners without the need for recertification. In this context, the collaboration between humans and robots occurs at the cognitive level, with workers and other non-experts designing and programming applications. This would not only empower shop floor employees to take over more

CONTACT Tudor B. Ionescu tudor.ionescu@tuwien.ac.at Human-Machine Interaction, TU Wien, Thresianumgasse 27, Vienna 1040, Austria

cognitive tasks but also provide robot programmers with access to the tacit knowledge of assembly workers (Ionescu 2019).

Drawing on the assumption that the notion of intuitive user interfaces is contingent on the user's prior knowledge and experience with technologies having similar user interfaces as well as with analogous day-to-day activities (i.e. conversations, text chatting, web searching, interactions with smartphones and tablets, etc.) it is hypothesized that intuitive robot programming is not a holy grail to be attained by one human-programmable interface (HPI) but a matter of providing diverse supports for that which is already known and experienced by the projected users. These supports may include app-oriented drag-and-drop features, voice-based programming, and other multimodal interactions.

This paper focuses on voice-based programming as one natural modality for supporting a more intuitive interaction with cobots. The proposed approach aims to close a gap in the commercial and research approaches to end-user robot programming by proposing a novel way of creating robot programs using voice commands. The main purpose of the proposed approach is to speed-up existing modalities of programming robots by (1) freeing the user's hands when teaching and finetuning cobot poses, and (2) endowing existing robot programming environments with programming by voice capabilities. In addition, the paper shows that the WSAPI provides a pragmatic, highly accurate (over 95% command recognition rate), zero-cost solution to voice command recognition for robot programming and provide a proof-of-concept, open-source implementation for speech-based robot programming, which is available online (https://assembly.comemak.at). The proposed approach thus aligns with current state of the art approaches in voice-based computer programming, which reduce conditions cause by physical strains (e.g. repetitive strain injury – RSI) and enable people with various disabilities to write code (Nowogrodzki 2018).

The novel contribution of this paper is threefold. First, it argues and demonstrates that programming cobots by voice has become more feasible and useful than ten or twenty years ago, when speech recognition systems did only reach an accuracy of around 80% in lab conditions. Second, it introduces a novel architecture for retrofitting existing robot programming environments by speech-based programming capabilities. Third, it provides an open-source implementation of the approach for a web-based, generic robot programming tool called *Assembly*. The novelty of the contribution consists in tackling speech-based robot programming rather than robot control – a domain, which has received relatively little attention in the multimodal and end-user robot programming literature in the past decade, as also noted by Villani et al. (2018). The significance of the contribution is represented by the pragmatic approach taken to implement speech-based cobot programming. The architecture introduced in section 2.3. can be used to endow existing robot programming tools with additional programming modalities, which are not limited to programming by voice. From an architectural perspective, the paper is significant because it introduces a novel way of connecting natural programming interfaces with existing robot programming environments event if those environments do not provide APIs or other means for supporting plugins and functional extensions.

The paper is structured as follows: It begins with a literature review of existing programming by voice approaches and principles for 'democratizing' cobot technology using simplified programming and multimodal human-robot interaction. Then, it introduces the programming by voice approach for a robot that provides a web-based HMI. This approach is then generalized to robots which provide conventional teach pendants and to a generic, web-based robot programming tool. Finally, the approach is evaluated based on three different implementations and selected usage scenarios.

1.1. Democratization of cobot programming and natural user interfaces

The use of cobots in industrial applications still falls short of enthusiastic expectations and forecasts. Whereas market surveys of the last years predicted an exponential growth and a vast distribution in industry, official cobot statistics of the International Federation of Robotics (IFR) note a total volume of 18.000 units, less than four percent of 2018's industrial robot sales worldwide (IFR 2019). Today's state of the art cobots largely rely on legacy programming concepts and user interfaces (UI) or started the implementation of function-block oriented graphical UI's that still do not fulfil users' expectations (Schmidbauer et al. 2020; Ionescu 2021; Piacun et al. 2021). Furthermore, most industrial UI concepts for cobot programming do not take into consideration enlarged user groups that span far beyond robot expert programmers and system integrators towards bystanders, industrial workers and laypersons who operate cobots in direct (safety-critical) interaction. That need is intensified by approaches to share tasks dynamically between cobots and workers (Schmidbauer et al. 2020a). Consequently, the notion of a cobot changes from an experts' system towards a manufacturing tool that can be used by literally everyone (Ionescu 2020b, Komenda et al. 2021). Therefore, one of the main motivations is to ease industrial engineering and manufacturing processes towards improved productivity and user-friendliness for an extended scope of potential users. In this context, democratizing cobot programming aims to provide a broader range of non-expert users with access to resources that empower them to use cobots effectively for their own (productive) purposes.

Human-machine interaction requires adequate interfaces to account for communication between human and artificial agents. The interaction usually covers both directions, from humans towards machines and vice versa. It also covers functionality, usability and user experience as core concepts. Since at least the latter largely refers to users' subjective feelings about the interactions with a system, the notion of *intuitiveness* currently guides the design and development of user interfaces. Intuitiveness within that context describes the ability that "[...] the users' unconscious application of prior knowledge leads to effective interaction" (Mohs et al. 2006). It usually refers to the individual, thus the 'particular user' and to the context of a 'certain task' (Naumann et al. 2007). The concept of interactive multi-modal robot programming refers to the combination of different interaction modes that can be combined seamlessly depending on their effectiveness as well as on user and task-specific criteria. Robot user interfaces evolved from Command Line Interfaces (CLI) over Graphical User Interfaces (GUI) towards Natural User Interfaces (NUI). Frameworks to combine gesture and speech control date back to the early 2000s (Perzanowski et al. 2001; Soshi, Paredis, and Khosla 2005) and are considered to be more favorable to the perception and evaluation of the robot (Salem et al. 2011).

1.2. Related work

This section reviews the relevant end-user robot programming tools and research results. First, it discusses state-of-the-art tools that are already on the market and current research approaches that are likely to be adopted by the industry in the coming years. Then, it discusses relevant speech-based programming approaches for industrial robots and computers, more generally.

1.2.1. End-user robot programming

End-user programming can be understood as programming by non-experts. To enable non-experts to program robots, suitable simplified programming models that depart from traditional text-based programming are required. Ko et al. (2004) note that, for end-users to be able to overcome learning barriers, simplified programming models should be more user-centric rather than computer centric, while fostering analogical reasoning and balancing abstraction and concreteness. In this sense, the same authors note that in a human-centric model, programs should be more similar to flowcharts and widgets (i.e. apps) rather than instructions, data, and lists (Ko et al. 2004). Drawing on these principles, current state-of-the-art end-user programming tools divide programs into *primitives* (sometimes called *behaviors*), *skills* (a specific combination of behaviors), *tasks* (programs or subroutines), and *groups* of other program elements (Pedersen et al. 2016). A primitive is a basic software function that controls a behavior of the robot, such as a joint or linear movement, an action of its end effector, and I/O communication. Skills are reusable automated handling functions that are built on a foundation of primitives. Skills are available from the tool vendor or third-party developers as plugins or apps that the user can customize. A robot task or program is composed of several primitives and skills that are organized in various ways based on visual representation, control flow, and data flow. Groups are program structures that correspond to loops, conditionals, and other generic control structures. Some of the programming models that are commonly used in graphical end-user robot programming tools include list-based, tree-based, flow-based, and block-based programming. In the following, a review of the relevant commercial end-user programming tools that are currently available on the cobot market are reviewed. Then, the relevant research approaches to end-user, multimodal robot programming are discussed. More comprehensive systematic surveys on end-user programming approaches can be found in (Ajaykumar, Steele, and Huang 2021) and (Villani et al. 2018).

1.2.1.1. Commercial end-user robot programming tools. Drag&Bot (2020) is a commercial, simplified, list-based, generic robot programming tool, which can be used from a web browser. Drag&Bot provides robot drivers for the following robot brands: ABB, Epson, KUKA, Yaskawa, Fanuc, Stäubli, Denso, and Nachi, and Universal Robots (both the CB and E-series). The tool integrates a web-based robot simulator in which entire robot stations can be modelled. The programming is performed by dragging, dropping, and configuring apps, called function blocks, with the help of configuration wizards. Drag&Bot provides a series of configurable program templates, which can be regarded as robot skills. Drag&Bot is the first web-based generic robot programming UI which can be used with an online, cloud-based account.

ABB Wizard is a simplified graphical programming UI, which builds on Blockly – the open-source block-based programming environment from Google. Several commercial industrial and non-industrial robots use Scratch and Blockly as their programming environment. The precursor of ABB Wizard is a research tool called CoBlox (Weintrop et al. 2018), which also uses Blockly as its main programming model. Wizard can be used with robots from the one-armed collaborative Yumi robot and the non-collaborative IRB 1100 robot. Wizard provides a block (i.e. app) library which can be used to program robots by dragging and dropping blocks onto the program canvas. Each block can be configured directly using inline parameters, or – in the case of motion blocks – by moving the robot with the joystick attached to the robot's teach pendant or by manually moving the robot. Users can also create their own custom blocks, called skills. A skill is a block-based program which contains several blocks, that are already available the library. The skill library can thus be extended by users or third-party developers by new skills through block composition.

ArtiMinds (Pieskä, Kaarela, and Mäkelä 2018) is a graphical, workflow-based, generic robot programming environment that borrows some features of block-based programming environments. ArtiMinds support different robot models from a series of vendors (currently, Universal Robots, KUKA, FANUC, ABB, DENSO, and Mecademic). ArtiMinds also provides templates and extensions for additional hardware like end effectors and cameras. ArtiMinds offers software modules that help to create a robot system composed of a specific robot, end effector, torque sensor, and camera. The functions of these systems can then be accessed and configured directly from corresponding ArtiMinds templates (i.e. predefined blocks that behave like apps or skills). To create a program, the user drags and drops blocks from a template library and then connects then via ports. Ports are assigned to input and output variables so that the outputs of one block can be connected to one or several other blocks. ArtiMinds also provides a simulation environment, which occupies the better part of the screen. Users can configure station layouts from predefined machine and part models, and test programs in simulation. ArtiMinds uses code generation to translate programs in robot-specific languages.

Wandelbots (Fitzek et al. 2021) developed a generic robot programming tool which provides a physical robot toolpath generation device, called 'TracePen'. This device allows users to teach robot motions by drawing paths in 3D space. Paths may follow the contour of parts that need to be treated in some way. The software uses machine learning to optimize the path and allows the users to adjust the manually generated path in an intuitive app.

1.2.1.2. Research approaches to end-user robot programming. RAZER (Steinmetz, Wollschläger, and Weitschat 2018) is a web-based generic robot programming tool, which supports programming by demonstration (through manual guidance of the robot) and skill-oriented programming. RAZER uses a list-based programming model that is similar to that of Drag&Bot. In RAZER, it is possible to program several robots with a single program. Programming is performed by adding and configuring skills in a similar fashion as with Franka Emika Desk. The provided skills support a limited number of different tool sizes and materials. RAZER also offers an expert mode, in which new, configurable skills can be added by experts to the tool either by implementing a state machine model called RAFCON or a custom interface. The skill model of RAZER thus resembles that of Franka Emika Desk.

The newest version of the tool (Steinmetz, Nitsch, and Stulp 2019) implements a semantic model for automatically detecting, instantiating, and configuring skills from the human-robot interactions performed by a user who programs the robot by demonstration—i.e. by guiding the robot to desired positions and manipulating the end effector. This reduces the time needed to program and configure tasks and skills since the user does not have to switch back and forth between the robot and the programming UI. The user's interactions with the robot are associated with a certain skill by using a time series of the robot's pose. A so-called semantic skill recognizer then matches this time series to a particular skill that is available in the tool's library and to also configure it automatically by extracting the position and rotation information from the time series data. To support skill matching, the skills are described using a so-called planning domain-definition language (PDDL).

iTaSC (Halt et al. 2018) leverages the process definition artefacts (symbols, notations, workflow language) from the VDI 2869 guideline. This guideline specifies a taxonomy of automating handling functions organized in three main categories: joining, handling, checking, adjusting, and special operations. Within each category, the guideline specifies a comprehensive set of operations, which are characterized by a symbol and description of the respective operation, including the parameters that are relevant to that operation. Production cells and processes can thus be specified symbolically as workflows of standardized operations. iTaSC uses constraint-based reasoning to generate a robot program consisting of a sequence of VDI 2860-conforming operations, called skills, from CAD models. These generated programs can then be further adjusted by non-experts.

Human Factory Interface (HFI) (Schäfer et al. 2021) is an experimental robot programming tool that provides CAD models of entire production cells in addition to robot and product models. These models are represented as a knowledge graph in a graph database and can be queried using SPARQL (an RDF query language). The resulting knowledge base, which is called 'world interface', can be used to comprehensively schedule, plan, execute, and reconfigure robotic production processes in a semi-automated fashion. HFI provides an intuitive web-based user interface in which users can define tasks in terms of their goals, which can be assigned to a robotic assembly cell. The UI provides information about the progress of running tasks and allows users to control the execution.

Semantic Mates is another ontology-based approach to semi-automated robot programming (Wildgrube et al. 2019). This constraint-based approach starts with CAD models of parts and uses a simple OntoBREP ontology that defines geometric entities and constraints. The ontological representation of Semantic Mates, which is stored in the knowledge base, is used to augment object models with additional information that is relevant to robotic assembly. Annotated object models can be used to program a robot to perform assembly tasks through simple drag-and-drop operations in the UI. Using the ontology, the UI supports the user in mating different work pieces together thanks to predefined geometric constraints. Once mated, the parts are forwarded to a component that maps the task of joining the parts together to the skills that are provided by the robotic work cell.

Robot telekinesis (Lee et al. 2020) is an approach that takes advantage of virtual object manipulation techniques that are used, for example, with 6 degrees of freedom virtual reality gaming controllers. The approach allows users to control and program robots by demonstration in a way that is not limited to collaborative robots. The telekinesis approach to programming by demonstration proved to be 4 times faster than using the teach pendant and just as fast as, yet physically less demanding than manual teach-in of the robot. Since the latter technique can only be used with collaborative robots, the telekinesis approach promises to make conventional robot programming by demonstration much more efficient.

Gadre et al. (2019) propose an end-user programming approach based on a VR headset connected to a Baxter collaborative robot. The user can define waypoints and end effector actions using their hands and fingers in space. The advantage of the mixed reality approach over the telekinesis approach is that the actions recorded in the program are displayed in an overlayed list, which can be edited by the user.

MEGURU is a tool that enables hand gesture-based end-user robot programming (Nuzzi et al. 2021). The tool provides support for an extensive set of one or two-hand gestures that can be used to control and program the robot. The system uses an off-the-shelf 2D camera. The authors argue that the gestures are easy to learn and that the system overall is easy to use. However, it is not clear what level of precision can be achieved using this modality of programming a robot. It is reasonable to assume that the achievable precision is below that obtained by using manual teach-in.

1.2.2. (Robot) programming by voice

Programming by voice or speech is a compelling idea, which received some attention by the software engineering and robotics communities in the past 25 years. Speech-based programming arguably helps to overcome so-called use barriers in end-user programming. According to Ko et al. (2004), use barriers 'are properties of a programming interface that obscure (1) in what ways it can be used, (2) how to use it, and (3) what effect such uses will have' (p. 3). Natural speech commands provide an alternative to manipulating non-intuitive interfaces, while enabling analogical reasoning.

From a technical perspective, while during the 2000s the main problem appeared to be the lack of reliable speech recognition techniques, owing to the recent advances in machine learning, today researchers are confronted with other problems, such as how to structure voice commands in order to produce code while minimizing the necessary corrections (Begel and Graham 2006; Arnold, Mark, and Goldthwaite 2000). In this sense, to reduce the speech recognition error, some researchers use code words instead of 1-to-1 mappings between what is being said/recognized and what is being coded. In the industrial robotics domain, programming by voice (i.e. producing program code rather than controlling or guiding a robot by voice) has received relatively little attention compared to other programming techniques, such as block-based (Weintrop et al. 2017), automated (Heimann and Krüger 2018) programming, or multimodal teach-in (Beschi, Fogli, and Tampalini 2019). More work appears to have been invested in controlling or guiding a robot using voice commands (Makris et al. 2014), whereby these commands are used to immediately call some pre-programmed function of the robot (e.g. different movements or more complex robotic skills).

By contrast, voice-based robot control is a well-known and researched problem in the industrial automation domain. In the 2000s the focus of this research was on the technology used to reliably recognize voice commands from users with applications in controlling mobile robots (Rogalla et al. 2002; Liu et al. 2005; Lv, Zhang, and Li 2008; Bugmann and Pires 2005). Thanks to the advances in the machine learning domain which helped to overcome the challenge of recognition accuracy, more recent approaches to voice-based robot control are focused on how to integrate wearable devices capable of speech recognition (e.g. Android smart watches (Gkournelos et al. 2018)) and how to interface voice-based control with existing robot application development environments (e.g. ABB Robot Studio (Pires and Azar 2018; Kumar et al. 2016)). Some of the newest approaches from the domain of human-robot interaction combine gesture and speech (Meng, Feng, and Xu 2020; Liu et al. 2018) or haptic and speech (Gustavsson et al. 2017) control into one systems while showing that a combination of the two yields better results in terms of accuracy and acceptance by users. Speech-based robot control has also been identified as one of the drivers of a new paradigm of 'symbiotic' human-robot interaction (Wang et al. 2019). In addition to these scholarly articles, a multitude of patents dealing with diverse aspects of robot control by voice have been published since around 2015, which shows that speech recognition has started to play an important role in industrial robot control and programming. Rogowski, 2013 presents a web-based remote voice control approach for robot cells, which allows the user to remotely issue complex commands which tell the robot to perform certain pre-programmed tasks (e.g. lifting, loading, unloading, picking, placing, etc.). The focus in Rogowski (2013) is on the grammar supporting the recognition of complex commands. The approach does not qualify as programming because voice commands are not used to generate robot code. Rather, they are used to parameterize and invoke pre-programmed tasks, while the user must engage continuously in a 'conversation' with the robot.

More recent approaches include a plugin for extending ABB's RobotStudio by speech-based robot control capabilities (Pires and Azar 2018). The plugin leverages the American English speech recognition and text-to-speech engines (TTS) provided with Windows 10 Pro. The system supports commands for controlling motors, running programs, and setting configuration options. However, it does not support commands for instantiating robot behaviors and skills within the text-based programs that can be created using RobotStudio.

As part of its RoboMaker service (Liu and Xu 2019), Amazon provides a service interface for connecting Alexa with robots through a ROS instance hosted in the cloud (i.e. robotic middleware as a service). The user triggers an Alexa skill using a voice command. If the command is recognized, it is converted into a machine-readable command, like a move command. The capabilities of the robot that can be controlled through voice commands are specific to each applications. Typically, users can trigger robot skills that are implemented in ROS.

Bingol and Aydogmus (2020) present a speech-based approach to controlling industrial robots based on voice commands, which trigger the execution of skills, such as drilling. The aim is to enable human-robot interaction with non-collaborative robots. In this approach, the robot provides feedback as to whether the command was successfully recognized and whether the robot is available to receive such commands. To enable speech recognition in the Turkish language, the authors developed a deep neural network, which is able to recognize commands with an accuracy of over 90%.

Kaczmarek et al. (2020) present an interesting approach to adjusting the cartesian position of a non-collaborative industrial robot arm using commands such as 'plus x', 'minus y', etc. This approach uses a C# library for speech recognition and qualifies as voice-based robot control rather than programming.

2. Cobot Programming by Voice: A Web Framework

This section describes a novel lightweight cobot programming by voice framework based on the new Web Speech API (WSAPI)—a W3C specification published and maintained by the Speech API Community Group supported by modern web browsers, such as Google Chrome, Mozilla Firefox, or Microsoft Edge. The framework explicitly targets the assembly application development phase, which usually requires many rounds of repetitive trial and error until robots reach the necessary precision, especially in small part assembly, such as PCB assembly. In this context, programming and testing cobot applications by voice can be more effective in terms of speed and difficulty. This would also empower shop floor employees to participate in the application development activities, as a growing body of work has called for (e.g. (Ionescu and Schlund 2019; Wilhelm et al., 2017)).

The proposed framework uses the WSAPI to wrap the programming structures used by a cobot produced by the Franka Emika company, which already offers a simplified web-based human programmable interface (HPI). In this environment, non-experts can program the robot by dragging, dropping, and configuring predefined apps in a sequence or according to simple patterns, such as repetitions of the same action or more complex composite apps, which can be acquired in an app store. This app-oriented way of programming cobots appears to be more intuitive than other industrial cobot HPIs because it draws on known patterns of interactions (e.g. with smart phones and tablets). This provides non-programmers with the opportunity of a fast and easy initiation into cobot programming. Nevertheless, the same programming technique becomes rather annoying and slow once a person is initiated into the craft of cobot programming. In this context, programming by voice can help to bridge the potential interest gap, which may arise when a person finds something to be too easy and limited. Also, and perhaps more importantly from an economic point of view, programming cobots by voice can boost the productivity of both beginner and expert programmers by allowing them to use their hands for guiding the robotic arm rather than inputting text or dragging apps and functions. The framework introduced in this paper thus aims to augment the multimodal programming capabilities of

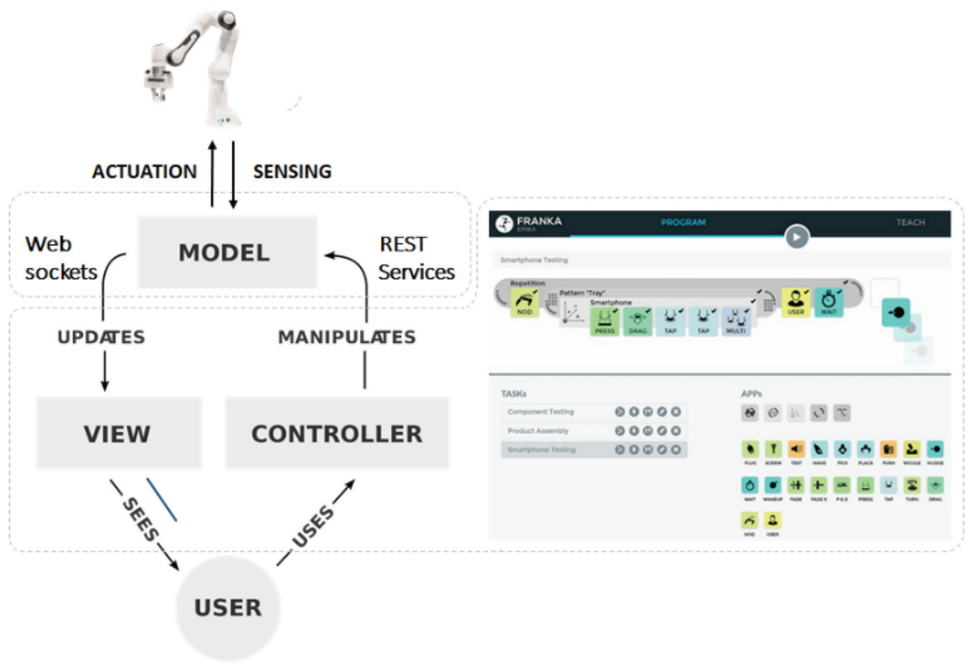


Figure 1. Architecture and screenshot of the Franka Emika Panda HPI.

existing cobots rather than providing a definitive replacement for any of them.

2.1. Programming the Panda Cobot: Features & Limitations

The Franka Emika Panda collaborative, industrial-grade robotic arm is one of the first of its kind to provide a web-based HPI. As depicted in Figure 1, The design of this HPI closely follows the model-view-controller (MVC) pattern (Leff and Rayfield 2001), with a REST and Websockets backend and a dynamic HTML5 & JavaScript web interface. The REST service components of Panda's HPI control and maintain the robot's state by processing sensor signals and actuating robot moves. The backend application, which implements the model in the MVC pattern, wraps the robot's native C++ API, called libfranka. This model maintains the current state of the robot (i.e. joint positions, applied forces, sensor values, actuation history, etc.) in memory. The web application, shown in Figure 1, is updated via web sockets to reflect the current state of the model. In this web environment, users can create new robot applications by dragging and dropping so-called robot apps from an app collection to a workflow pane to form a sequence of basic actions, whereby each app in the sequence represents a configurable instance of a movement or behaviour of the robot (e.g. taught-in or relative moves, grip-per actions, waiting for user input, etc.). In addition, there exist so-called pattern apps, which implement predefined applications patterns (e.g. picking and/or placing parts in a tray, or more advanced program structures). Named app sequences, which are called tasks, are persisted in the model, whereby all user interactions generate calls to the REST services. This app-oriented HPI is coupled with a series of buttons available on the robot's head, as shown in Figure 1. Two of these buttons are used to move the arm in free-drive mode. The other ones are used within the wizard-like configuration of each app, which is activated by clicking on the respective app, as shown in Figure 1. This way the user only needs to interact with the laptop or tablet when dragging and dropping a new app into the workflow. For most of the available apps, all other parameters can be configured using the buttons from the robot's head. In practice, however, users prefer to set certain parameters and click through the wizards using the mouse or touch screen of the laptop of tablet because the buttons are relatively hard to press and thus induce physical strains into the wrist when used repeatedly for a long time. Also, configuring the apps using the buttons is usually slower than using the mouse and keyboard. Hence, while Panda's HPI arguably represented a leap in intuitive robot programming, it has some obvious limitations when it comes to ergonomics and speed.

In our experience with using this robot in research and teaching activities, these limitations not only reduce the productivity of users, who need to switch back and forth between the robot's head and the computer but may also pose a safety hazard when more than one individual works with the robot. Students, for example, tend to divide tasks so that one person will move the robot, while another one configures the apps. At times, however, both persons may stare at the computer screen, whereby one of them (the one who guided the robot's head) may be in close proximity to the robot, which – on occasion – may perform unexpected moves (e.g. in case of a software or hardware error) and thus potentially harm users. When working in teams of two, only one person should program the robot, while the other one should watch that all safety-related rules are being followed (Ionescu and Schlund 2019). These roles should be switched periodically.

2.2. Using the Web Speech API for Programming by Voice

The WSAPI is a collection of client-sided JavaScript functions embedded in modern web browsers (notably Chrome and Firefox), which allows web developers to leverage the newest machine-learning based speech recognition technologies. Common use cases include voice web search, speech command interface, speech translation, dialog systems, multi-modal interaction and search, etc. The WSAPI can access the computer's microphone (or any other microphone connected to it) via the browser. To ensure privacy, the user is asked for permission before listening is enabled. As opposed to older speech recognition frameworks, the WSAPI supports a wide variety of languages and has a very high recognition success rate. This makes corrections only necessary in the presence of external acoustic interferences, provided that the user is a fluent speaker in his/her language of choice.

Once enabled, the WSAPI listens to what is being said and converts spoken sequences into words placed, for example, in an array. To implement cobot programming by voice, a script that can be injected into Panda's web-based HPI using the 'bookmarklet' technique was developed. A bookmarklet is a bookmark containing JS commands stored in the browser's 'favorites' bar, which adds 'one-click' functionalities to the currently displayed web page. This technique has been effectively used to embed or extend programming environments in existing webpages. The current approach uses the same mechanism to inject JavaScript code into Panda's HPI, which automates the instantiation, loading, configuration, and execution of apps and tasks in the robot's native HPI according to the user's voice commands.

The bookmarklet technique was used to extend Panda's HPI by voice programming falls into the broader category of graphical user interface (GUI) automation, which is primarily used in software testing (see (Ionescu and Schlund 2019) for details). Yet, with the increasing diversity and complexity of software used in companies and privately, more GUI automation use cases emerged in the past few years. In this spirit, the proposed framework aims to bring this new philosophy of interfacing with existing software GUIs to the world of collaborative robotics programming to the end of boosting productivity.

In the following, some technical details about the implementation of the framework before turning to the empirical evaluation and discussion of the approach are discussed.

2.3. Framework Design and Features

Figure 2 illustrates the architecture of the 'Programming by Voice' extension to the Panda robot's HPI.

Building on the robot's HPI, which implements the MVC design pattern, the user injects a so-called 'Programming by Voice' (PBV) controller into the current browser page (i.e. the robot's HPI shown in Figure 1). Once injected, the PBV controller activates and maintains an open connection to the Web Speech API embedded in the browser, which enables it to listen for voice commands from the user. Upon recognizing one of the commands listed in Table 1, the PBV controller may use the visual controls available to users by automating the user actions needed to accomplish certain tasks (e.g. to drag, drop, or configure different robot behavior apps). The PBV controller can also directly manipulate the server-side components of the system by sending REST

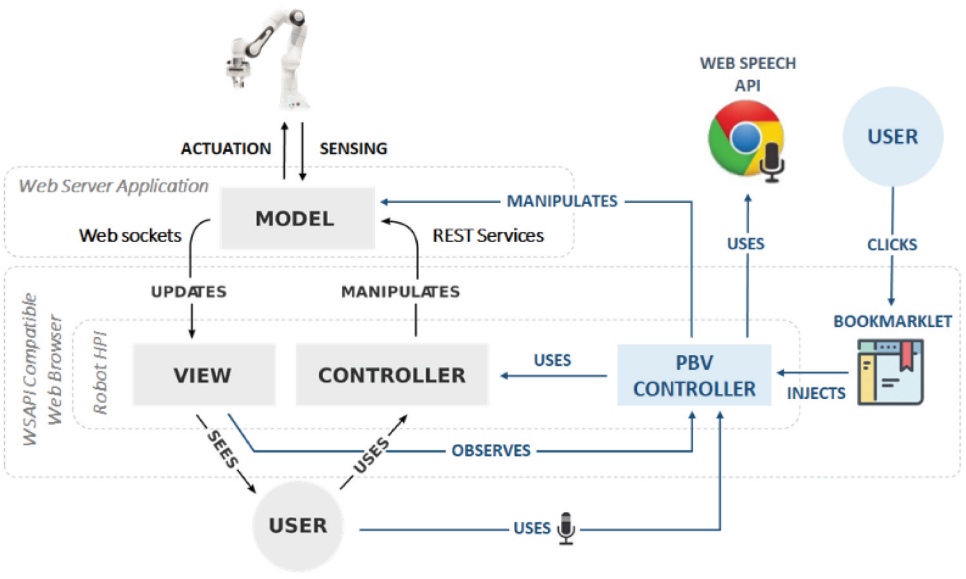


Figure 2. Architecture of the "Programming by Voice" extension.

Table 1. List of supported voice commands (GRP=Group, CMD=Command).

GRP	CMD	Functionality
1	Speed/Force/ Load/Speed-up <#>	Sets the velocity/grasping force/gripper load/acceleration of the arm for all subsequent apps to the specified integer value <#>. A subset of these parameters are required by the movement and gripper apps.
2	Grasp	Initializes the "Gripper Grasp" app which allows users to manually set the opening of the gripper so as to grasp a certain object. After triggering this app, the users are able to use the buttons on the robot's head or to manually set the gripper fingers to the desired position.
	Hand	Initializes the "Gripper Move" app which allows users to set the opening of the gripper using the buttons on the robot's head or by manually moving the gripper fingers.
	Motion	Initializes the "Cart Motion" app, which allows users to teach in a movement of the arm using several waypoints. Waypoints are set by driving the robot's arm to the desired pose and then pushing the OK button on its head.
3	Okay	After having set all the parameters of an app by using the buttons on the robot's head and/or manually moving the robot's arm and fingers, this command will auto-complete all the information requested in the subsequent dialogues of the app-specific wizard (e.g. velocity, force, acceleration, load).
	Start	This command starts the execution of the task being programmed.
	Stop	This command stops the execution of a task.

service calls. On the server side, the application model (which contains the robot's state, programmed tasks, and app implementations) is maintained and persisted in a database. The robot's HPI is thus augmented by a new programming mode, which enables users to create tasks while keeping their hands free (e.g. for teaching positions by guiding the robot and configuring the gripper using both hands). This eliminates the need for switching back and forth between the robot and the laptop or tablet running its HPI. The following commands are currently supported:

The first group of commands can be issued at any time while working with the robot's native HPI. These commands save time and clicks by allowing users to set reusable parameters before configuring specific apps. While configuring an app, which requires certain parameters, issuing the 'Okay' command will automatically set the values of the app-specific parameters to those of the matching global parameters. If users forget to set these global parameters to the desired value, they can still be changed manually at a later time using the app's configuration wizard.

The second group of commands instantiate apps for actuating the robot's arm and tool (e.g. gripper). In the robot's native HPI, these off-the-shelf apps need to be dragged from the app library and dropped in the workflow pane of the currently active task. The task of instantiating apps, which can take 5–10 s per app, can be automated using voice commands. This saves time because, while the robot's HPI performs the initialization of a task, the user can already drive the robot into the desired pose for the first waypoint of the motion or set the width of the gripper fingers. After all waypoints are configured using the buttons on the robot's head and/or the gripper width is manually set by moving its fingers, the user can issue the 'Okay' command to trigger the auto-completion of the remaining app parameters, such as the velocity and acceleration of a movement and the force and load for pick and place operations.

The commands from the third group are used to trigger actions, which would require one or several mouse clicks or tapping the touch screen of the device running the robot's HPI (e.g. laptop, tabled, etc.). The 'Okay' command triggers a context-dependent series of actions so as to autocomplete the parameters of the app being configured and to automatically click-through all remaining dialogues of the respective app configuration wizard. The 'Start' and 'Stop' commands will simply start and stop the current active task. By using voice commands the user can focus on interacting with the robot through haptic means. To switch from the task programming into the task testing and running mode, the user needs to pull up the robot's safety stop button, which is pushed down during teaching and programming. Hence, using the start/stop commands, users can save time, for example, by being able to focus on the robot's actions during testing while keeping the safety stop button at hand. This also contributes to a safer interaction with the robot during testing, since the user's attention is not distributed between the robot and the programming interface.

3. Generalization of the approach

This section discusses two approaches to generalizing the proposed speech-based programming approach by (1) adapting it to the case of human-machine interfaces (HMIs) that are hosted on conventional robot teach pendants, and (2) integrating it into an open-source, block-based generic robot programming tool, called *Assembly* (https://assembly.comemak.at). The aim of this section is to lay the groundwork for the evaluation of the approach and to show that the WSAPI is a versatile tool for speech-based programming that can be integrated into multiple, existing cobot programming environments with relatively little efforts. The generalization of the approach also substantiates the claim that it helps to democratize cobot programming more generally.

3.1. A generic meta-controller architecture for supporting additional human-robot interaction modalities

Currently, teach pendant software can only be extended through officially supported channels, such as plugins or APIs. Some robot vendors have created software ecosystems and app stores (e.g. Franka World, UR+, ABB Robot Apps, etc.), where third party developers can develop and offer their plugins. Plugin interfaces, however, are still limited to integrating new features into existing HMIs, such as new function buttons or apps for various end effectors and sensors. Plugin interfaces do not provide support for extending the programming modalities of a robot by third party developers. One way of extending the programming modalities of a robot without losing the existing features of the vendor-provided teach pendant and HMI is to implement a meta-controller, such as the PBV controller for Desk, which emulates the interactions of a user with the HMI that are required to program the robot. Such a meta-controller can be implemented using tools and techniques from the domain of GUI automation (Yeh, Chang, and Miller 2009).

The overarching goal of the meta-controller approach can be expressed as follows: When developing new features for the HMI of an industrial robot, which go beyond the functionalities supported by official APIs and plugin (eco)systems, it is desirable to implement and deploy the new features on top of the vendor-provided HMI of the robot in a non-intrusive way without blocking, overwriting, replacing, or otherwise interfering in a nonbeneficial way with the vendor-provided features of the HMI. To achieve this goal, the HMI of the industrial machine can be extended by a meta-controller consisting of a remote view, a condition monitor, and an HMI object (see Figure 3). The remote view connects through a remote connection interface provided by the HMI to enable remote viewing and control of the HMI by the condition monitor and the HMI object. The condition monitor observes the graphical elements of the HMI in the remote view and sends an interaction command to the HMI object whenever a visual state,

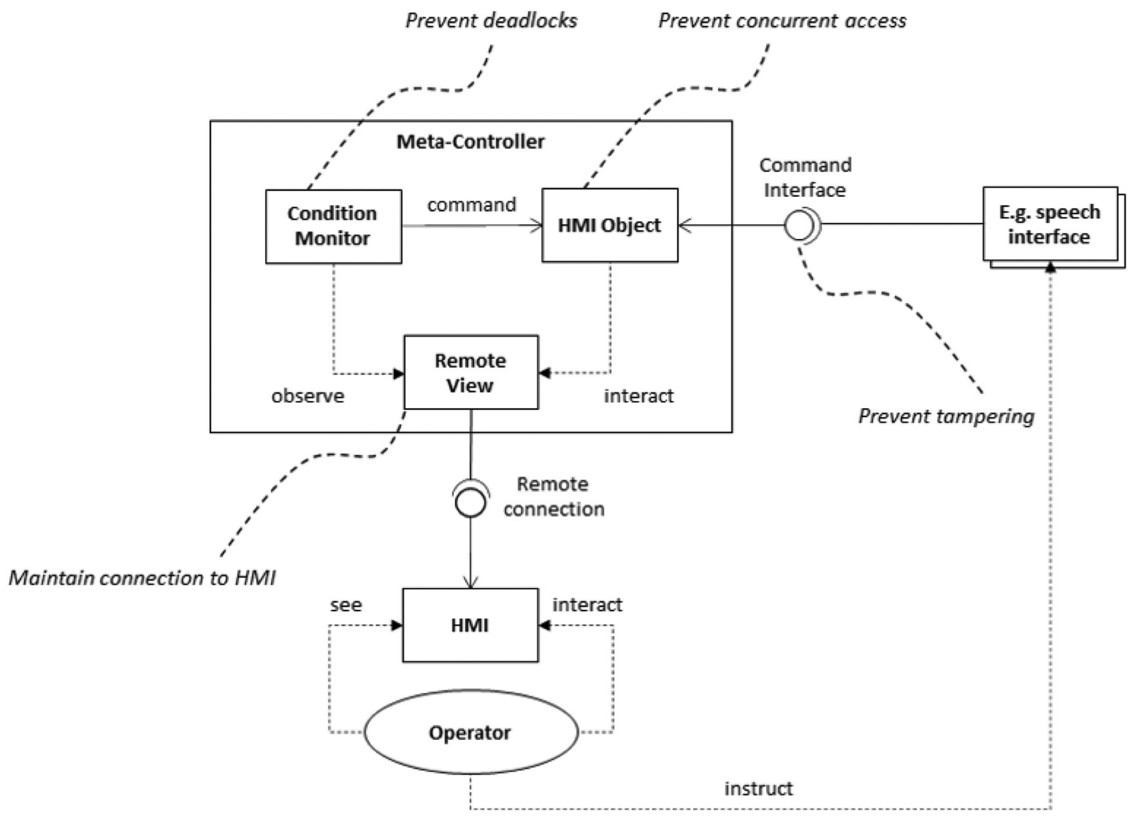


Figure 3. Generic meta-controller architecture.

which is designated to trigger such an interaction is detected. Upon receiving a command, the HMI object performs a corresponding interaction in the remote view, which changes the state of the HMI. A command interface is provided to facilitate user interactions with the HMI through natural interfaces, such as speech or gesture recognition. To prevent tampering with the command interface and the meta-controller device, in the case of robots, physical protections can be used. The meta-controller must prevent deadlocks in the condition monitor by maintaining the state of the HMI. Also, concurrent accesses to the HMI object must be prevented by queueing commands. The meta-controller must maintain the connection to the HMI using a watchdog.

Figure 4 provides a detailed view of the components of a generic meta-controller. The condition monitor is multi-threaded, with each thread monitoring a specific condition in the replicated HMI. The conditions are defined using graphical patterns that are recognizable in the HMI. Conditions can be defined based on the appearance and disappearance of various graphical and textual elements in the HMI. To prevent deadlocks, the condition monitor maintains a consistent state of the HMI, which is updated by the corresponding thread whenever a condition is met. In addition, the conditions must be defined in a mutually exclusive way such that only one condition can be met at a time. Whenever a condition is met one or several commands are issued by the concerned monitoring thread and sent to the HMI object.

The HMI object uses the same means as the operator to interact with the machine at the human-machine interface, for example, pointer devices, keyboard inputs, or tapping. One condition change detected by the condition monitor can lead to one or several interactions being automatically performed by the HMI object. The HMI object is similar to a Page Object (Leotta et al. 2013), which is used in the UI testing domain. In addition to the Page Object, which provides page or panel wrappers, the HMI object provides a concurrent command queue to prevent concurrent access to the input devices if commands are received from the condition monitor and the command interface at the same time. The HMI object

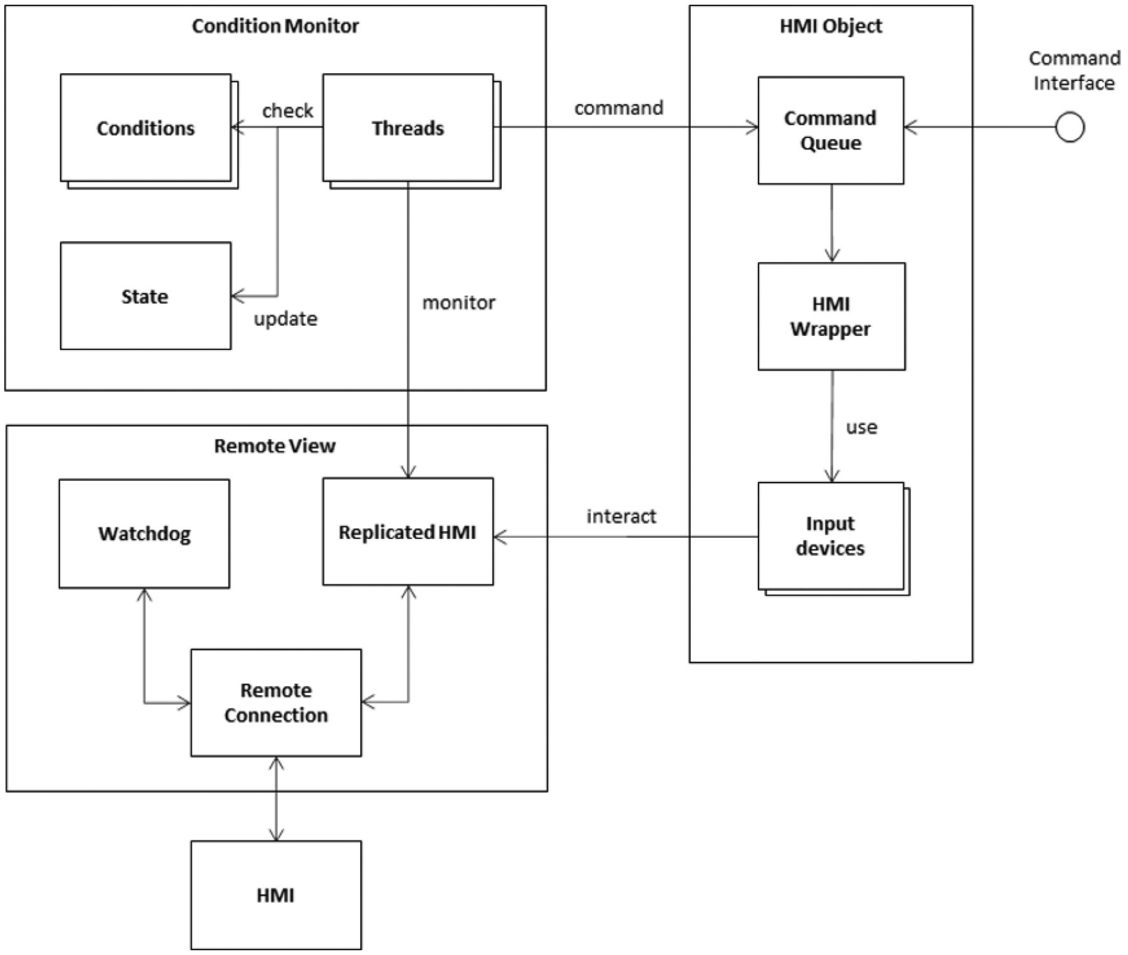


Figure 4. Detailed view of the meta-controller architecture.

wraps the HMI with an application-specific API, which allows clients to manipulate the HMI without explicitly issuing click, type, and other low-level input commands. The wrapper provides interaction methods that correspond to commands received from the condition monitor or through the command interface. The scope of the interaction methods should be small enough to facilitate reusability and broad enough to encompass a number of cohesive interactions (i.e. click on a button, then input some numbers, and then click ok to finalize the interaction). The interaction methods provided by the HMI object are atomic in the sense that they either complete successfully or they rollback the state of the HMI before issuing an error. Ensuring atomicity and handling errors are responsibilities of the HMI object.

The remote view replicates the HMI of the industrial automation system in a virtual screen which is accessible only to the condition monitor and the HMI object. The remote view manages the remote connection to the HMI to ensure maximum availability, e.g. using a watchdog that monitors the connection and attempts to reconnect in the case of a breakdown.

A meta-controller maintains relations both to the human operator and to the HMI, thus acting as a second operator who assists the human operator. A Meta-controller assists the human user of the system in an intuitive way, without creating confusion. For example, it can provide additional features, which the native HMI of the system does not, e.g. a role-based access model, error handling, execution monitoring and logging, programming and configuration wizards, etc. A Meta-controller reports back to the user either silently by performing as instructed or through popup and other kinds of messages in the HMI. In addition, a Meta-controller facilitates the non-intrusive integration of natural interfaces, such as speech and gesture recognition, which can be instructed by the human operator to perform interactions in the HMI that would have otherwise required manual inputs by the operator. The command interface can also be used by other applications to control and configure the machine in the absence of a human user.

Currently, all major robot vendors provide commercial solutions for remotely viewing the HMI of the teach pendant, e.g.:

- KUKA – Virtual Remote Pendant (VRP)
- ABB – FlexPendant (RobotStudio® provides us a virtual teach pendant)
- Universal Robots – RealVNC over an Ethernet connection
- YASKAWA – Remote Pendant Display
- FANUC – Remote iPendant

3.1.1. Speech-based programming for universal robots

In this example, the Universal Robot 5 (UR5) from the CB series is considered. The UR5 is a cobot that provides a conventional teach pendant that implements a tree-based simplified programming model. UR5's HMI is easy to use but the programming is ineffective because of the need to continuously switch between menus and between the teach pendant and the robot, when teaching waypoints (Ionescu et al. 2020; Ionescu 2020a). The vendor provides an API, which can be used to develop plugins for UR5's HMI. However, these plugins can, for example, extend the existing function library by adding a new configuration window. Existing features, such as teaching waypoints, cannot be extended, or otherwise modified by third-party developers through plugins. Hence, in Universal Robots' software ecosystem, called UR+, there are no plugins that enable the extension of the programming and interaction modalities that the cobot currently offers (i.e. via the teach pendant and direct manipulation of the robot's pose). This example demonstrates how UR5's HMI, called Polyscope, can be extended by voice-based programming using the meta-controller approach. The required system architecture corresponds to that from Figure 5.

To implement the voice-based programming extension, the SikuliX (http://sikulix.com) GUI automation tool was used. SikuliX is a Java-based tool for performing automated software tests of graphical user interfaces (GUIs) using screenshots of the software to be tested. SikuliX provides a simple IDE (integrated development environment), in which the visual patterns that are used to identify the key elements of a GUI are displayed in line with Python code. SikuliX is comparable to GUI automation software such as Selenium or AutoHotkey and like these can be used to meta-control websites or application software in any operating system. Testing and remote control of other devices are also possible via a simulator or Virtual Network Computing (VNC). In the context of the architecture from Figure 5, SikuliX was used to implement the meta-controller, which operates on the replicated HMI of the UR5 robot. The replication or mirroring of the HMI was realized using the RealVNC (https://www.realvnc.com) remote

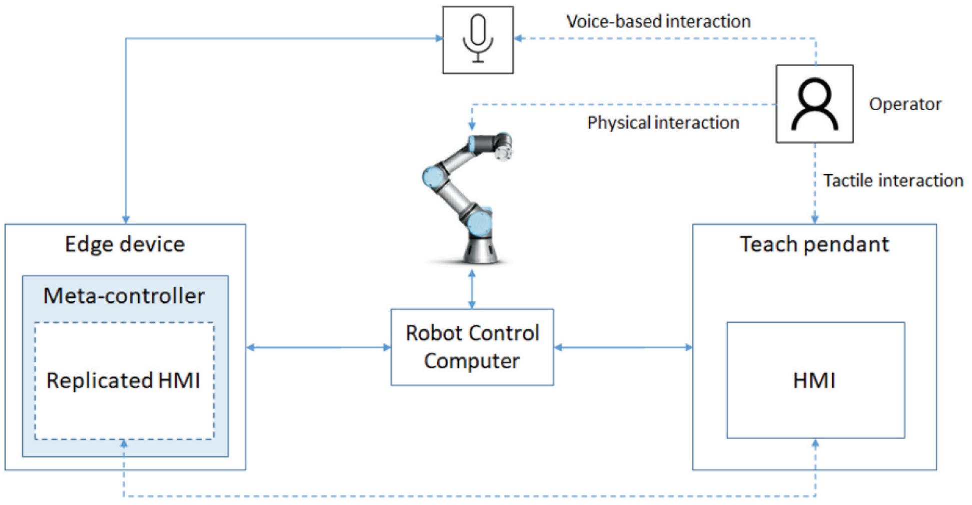


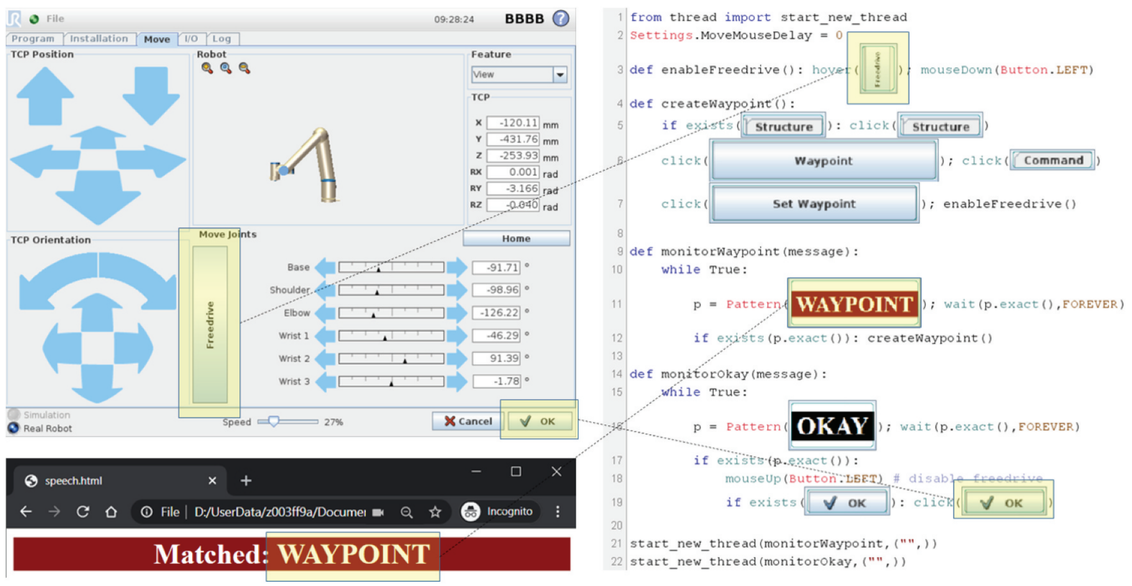
Figure 5. Speech-based robot control and programming for a Universal Robot using a meta-controller hosted on an edge device.

viewing and control software. A prerequisite for using this architecture is for the robot to support such a remote connection to the teach pendant. This feature is provided by many robot vendors because it is useful in the context of remote support. The COVID-19 crisis also increased the use of remote robot control due to the access restrictions imposed by governments. To enable the remote connection to UR5's teach pendant, the RealVNC server component needs to be installed on the teach pendant by connecting to it via SSH over an Ethernet connection. Once installed, it is possible to replicate the robot's HMI on another computer in a window. A tool like SikuliX is capable of monitoring and controlling the replicated HMI by emulating the user interactions with the HMI that are required to perform any task that is supported by Polyscope.

The UR5 robot teach pendant provides a 'Freedrive' button that enables the user to manually drag the robot to a desired position. However, the user must keep the button pressed while moving the robot, which occupies one hand all the time. In addition, many users complain that keeping the Freedrive button pressed all the time with one hand, while manipulating the robot using the other hand is cumbersome and induces physical strains in the wrist. More ergonomic solutions include, for example, a mountable flange ring having a button, which activates the robot's Freedrive mode. Using the meta-controller approach, a simpler and inexpensive alternative can be implemented using speech-based commands. To demonstrate this, the web-speech API was used to recognize the following commands: 'Waypoint' – adds a new waypoint to the current program by emulating the necessary HMI user interaction workflow and activates the free drive mode until the 'Okay' voice command is issued by the user, upon which the free drive mode is deactivated and the waypoint is set.

Figure 6 illustrates the implementation of this mechanism in SikuliX. The SikuliX program monitors these commands in separate threads and performs the necessary user interaction workflows for creating a waypoint, activating the free drive mode by keeping the corresponding button pressed and for deactivating it and storing the waypoint. When the free drive mode is activated, the user can move the robot to the desired position. This position is stored when the 'Okay' command is issued. Hence, the user can store multiple waypoints by repeating a simple three-step workflow, which saves about 50% of the time needed to accomplish the same operations manually.

Like in the case of the Franka Emika Desk HPI, in the case of the Universal Robot, the WSAPI was used to implement a simple speech enabled web page, which recognizes the two commands. At the same time, a remote connection to the UR5 robot's HMI is maintained on the same screen next to a browser window in which the WSAPI listens for commands, as shown in Figure 5. The WSAPI is configured to match the two



```
from thread import start_new_thread
Settings.MoveMouseDelay = 0
def enableFreedrive(): hover( Freedrive ); mouseDown(Button.LEFT)
def createWaypoint():
    if exists( Structure ): click( Structure )
    click( Waypoint ); click( Command )
    click( Set Waypoint ); enableFreedrive()

def monitorWaypoint(message):
    while True:
        p = Pattern( WAYPOINT ); wait(p.exact(),FOREVER)
        if exists(p.exact()): createWaypoint()
def monitorOkay(message):
    while True:
        p = Pattern( OKAY ); wait(p.exact(),FOREVER)
        if exists(p.exact()):
            mouseUp(Button.LEFT) # disable freedrive
            if exists( OK ): click( OK )

start_new_thread(monitorWaypoint,("",))
start_new_thread(monitorOkay,("",))
```

Figure 6. Speech-based robot control and programming for a Universal Robot using a meta-controller hosted on an edge device.

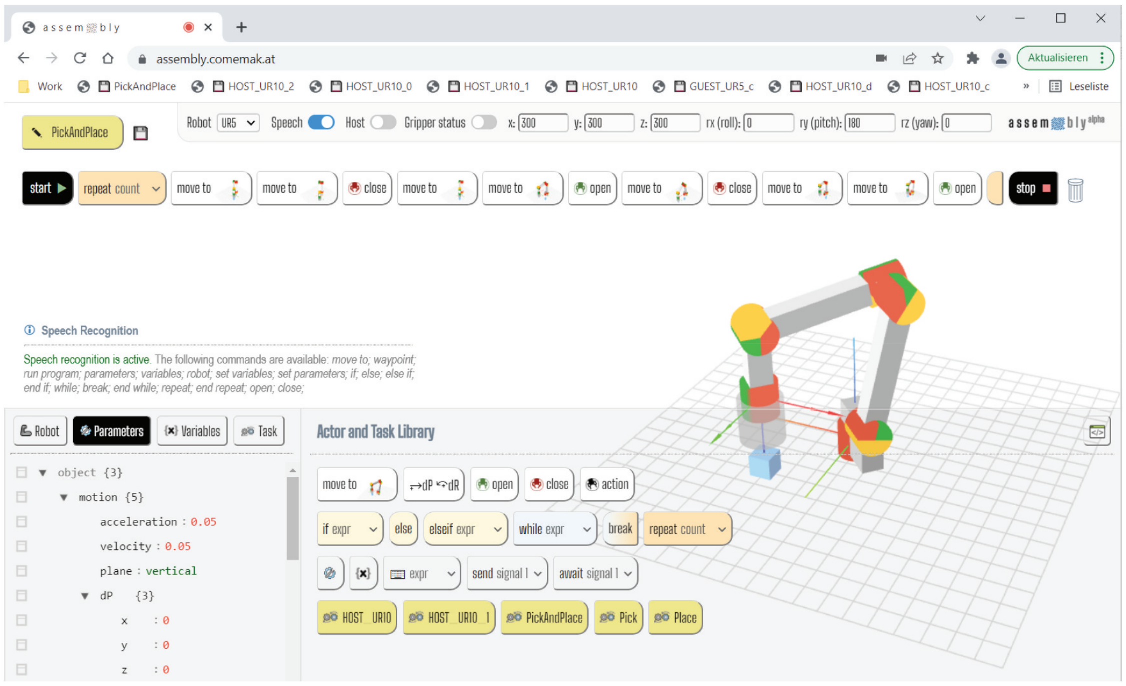
commands and to display a large banner with the matched command for three seconds. To handle the speech commands, a condition monitor called 'monitorWaypoint' in Figure 5 was created. When this pattern is recognized on the screen region of the HMI, the condition monitor calls the 'createWaypoint' function, which emulates the user interactions that are necessary to create the waypoint in the UR5 program. At the end of this function, the meta-controller is instructed to keep the freedrive button pressed, until the user issues the 'Okay' command. During this time, the user can use both hands to move the robot to the desired position.

Note that in this case, the HMI automation engine concomitantly monitors two different GUIs (i.e. the browser window and the UR5 HMI). The meta-controller, which was implemented using SikuliX, uses 22 code lines to recognize the labels corresponding to the voice commands in the browser window and to emulate the corresponding user interactions in the HMI. Additional voice commands can be implemented in a similar way, e.g. for speeding up robot programming and enabling people with different forms of impairment to operate robots.

3.2. Integrating speech commands into block-based robot programming environments

Assembly is a web-based, block-oriented generic robot programing environment, which was conceived as a combination between Franka Emika Desk and Blockly. From Desk it adopts the idea of modeling robot programs as a sequence of configurable blocks or apps (which in *Assembly* are called actors) in a way that (1) reflects the sequential nature of robot actions and (2) does challenge novice users, who have little or no programming experience. From Blockly, it adopts the idea that a simplified programming environment should also enable users to write complex programs, which include conditionals and other control structures, which is currently not possible in Desk. Providing all the features of a textual programming language in a simplified way also amounts to what the developers of Blockly call an 'exit strategy' (Fraser 2015) from the world of simplified programming to that of text-based programming.

Assembly uses the approach described in section 3.1. to enable generic robot programming by generating code in third-party robot programming environments, like Polyscope, using so-called code



Speech Recognition

Speech recognition is active. The following commands are available: *move to; waypoint; run program; parameters; variables; robot; set variables; set parameters; if; else; else if; end if; while; break; end while; repeat; end repeat; open; close;*

```
object {3}
    motion {5}
        acceleration : 0.05
        velocity : 0.05
        plane : vertical
        dP  {3}
            x  : 0
            y  : 0
            z  : 0
```

Figure 7. *Assembly* tool with voice commands enabled.

Table 2. List of supported voice commands in *Assembly*.

Command	Functionality
move to/waypoint	Adds a "move to" actor to the workflow, which memorizes the current pose of the simulated robot. When running the program, the robot will move to the specified waypoint.
open/close	Adds an "open"/"close" actor, which opens or closes the gripper, respectively.
set variables/set parameters	Adds a "set variables"/"set parameters" actor to the workflow, which captures the current values of the variables or parameters, respectively. These values will be used by all subsequent actors in the workflow until another set variables/set parameters changes them.
if/else if/end if	Adds an "if"/"else if"/"endif" actor to the workflow. This enables users to program conditional execution.
repeat/end repeat	Adds a "repeat"/"end repeat" actor to the workflow.
while/end while	Adds a "while"/"end while" actor to the workflow.
break	Adds a break actor that—when used inside of a repeat or while loop—breaks the respective loop.
robot	Opens the robot pane, which shows the cartesian position and joint angles corresponding to the current pose and model of the simulated robot.
parameters	Opens the "Parameters" pane, which allows users to set various actor and task parameters.
set <parameter name> <value>	Sets the value of the specified parameter.
variables	Opens the "Variables" pane, which allows users to create and set the values of various parameters, which are used with control structures and other actors.
set <variable name> <value>	Sets the value of the specified parameter.
delete	Removes the last actor in the sequence.
run program	Executes the current program.

generators, which build on GUI automation tools like SikuliX. The approach to generating robot programs from *Assembly* is described in more detail in (Ionescu et al. 2020).

To enable voice commands in *Assembly*, the WSAPI was integrated into it and the range of supported commands was extended compared to the Desk implementation. Figure 7 illustrates the *Assembly* environment, in which voice commands have been enabled. Programming in *Assembly* is straightforward: just like in Desk and Blockly, the user drags and drops actors from the 'Actor and Task Library' into the program workflow anywhere between the *Start* and *Stop* blocks. At the same time, the user can position the generic 6-DOF robot by dragging its end effector (a simulated suction gripper) or by adjusting the coordinates using the input fields from the top of the program workflow. To speed up the programming task, the user can enable voice commands, which frees the mouse and keyboard and allows the user to position the robot and focus on the next task. Table 2 explains the supported commands.

As opposed to Franka Emika Desk, *Assembly* also supports instantiating more complex program structures like loops and conditionals using voice commands. A*ssembly's* blackboard architecture allows the user to configure the actors using the *Parameters* and *Variables* panes. This architecture allows users and services to post all information that is relevant to the successful execution of the program (i.e. parameter and variables) onto a centralized so-called blackboard, which is implemented as a JSON object. This contrasts Desk's approach, in which each app maintains its own set of parameters. In *Assembly*, all available variables and parameters can be set using the so-called 'set parameters' and 'set variables' actors. Once set this way, all subsequent actors in the program workflow will use the snapshot of the values of the variables or parameters that were previously set using one of the set parameters/ variables actors. This allows reusing some of the parameters across multiple actors as well as setting the values of parameters and variables using voice commands. Concerning program control structures, *Assembly* uses a C-style rather than a Python-style model, which requires closing control structures using a terminator (i.e. an accolade in the case of C/C++ and other C-style languages). In *Assembly*, terminators are control structure specific and can be added using the 'end <control structure>'; commands (e.g. 'end while').

4. Evaluation of the approach

This section presents an evaluation of the WSAPI-based cobot programming by voice approach based on the three different implementations introduced in this paper. The ability to generalize the approach to various third-party robot programming tools also speaks for its validity. In this context, this section provides some concrete usage scenarios in which programming by voice can help to speed up programming while improving concentration and reducing physical strains.

4.1. Franka Emika desk

The evaluation of the programming by voice extension of the Franka Emika Desk tool was conducted on the basis of two application scenarios. In these scenarios it is assumed that shop floor employees (e.g. assembly workers and planners) engage in programming the robot in the following situations: (1) Fast programming of pick and place operations for small lot sizes using a basic teach-in procedure. (2) Limited modifications brought to a robot program in the context of an advanced robotic automation application, which already provides means for autonomous manipulation of work pieces and error recovery in case of collisions, whereby the envisioned error recovery process partly requires human intervention.

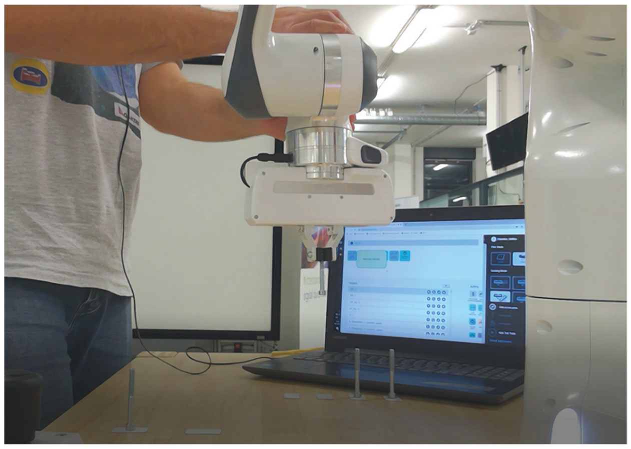


Figure 8. Setup of the pick-and-place application used in the evaluation in the TU Wien Industrie 4.0 pilot factory in Vienna, Austria.

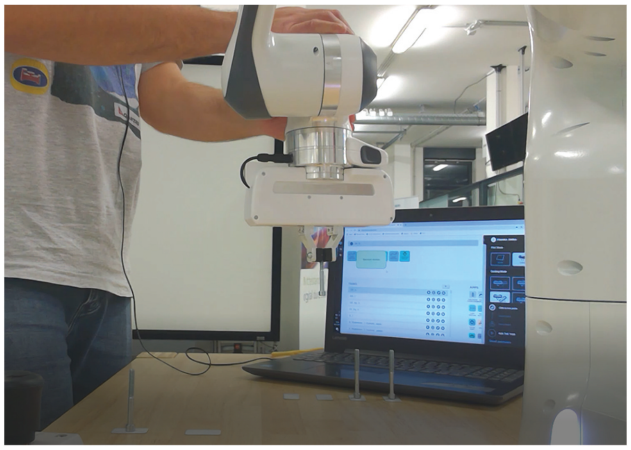


Figure 9. Setup of the pick-and-place application used in the evaluation in the TU Wien Industrie 4.0 pilot factory in Vienna, Austria.

Whereas in the first scenario, the focus will be on the ease and speed of programming a cobot, the second scenario illustrates how already programmed and possibly certified applications can be modified during productive use by shop floor employees without interrupting production for a long time.

4.1.1. Scenario 1: faster cobot programming

Figure 8 shows the setup of the simple pick-and-place application. The evaluation was conducted with the help of four human participants, who did not know how to program the Panda robot before. The participants were first introduced in how to use Panda's native HPI and then asked to manually program the pick-and-place task depicted in Figure 9 using the robot's HPI, which was running on a laptop. The participants used a mouse and the laptop's keyboard to program the task, which consisted in picking and placing four screws. Then, the same participants were introduced to the programming by voice extension to the robot's HPI and asked to program the same task without using the mouse and keyboard. Video 2 linked from this webpage (https://blockly-desk.comemak.at/demos/code/speech.html) demonstrates how this task is performed using voice programming. Figure 10 shows a breakdown of the operation durations with respect to the three groups of supported voice commands and their mouse and keyboard equivalents. In addition, the time spent manipulating the robot and waiting for the robot's HPI to react were also considered in the evaluation. These results show that, for the tested scenario, programming by voice combined with the haptic manipulation of the robot was around 46% faster than using the robot's native HPI, whereby most of the time was saved by not having to manually set app parameters and to click through the app configuration wizards.

This scenario arguably demonstrates the benefits of using programming by voice instead of manual wizard-based configuration of apps for teaching pick and place tasks. The main benefits are in terms of time savings, safety, and ergonomics.

4.1.2. Scenario 2: error recovery through human intervention

The second application scenario consists of a chess-playing program, which is able to recover from an error by requesting human intervention. If the robot

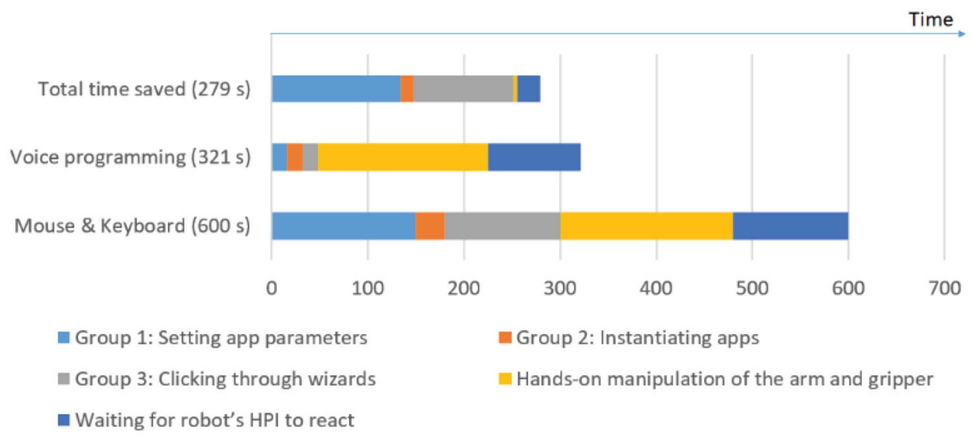


Figure 10. Breakdown of operation durations in scenario 1.

fails to grasp a piece due to a collision, the program stops and waits for human intervention. In preparation of this intervention, the program automatically creates a new task, which opens the gripper wider in order for the robot to grasp the piece correctly. The human is only required to set the width of the gripper so that it will be able to grasp the piece without disturbing any other pieces on the board. Video 2 linked from this webpage (https://blockly-desk.comemak.at/demos/code/speech.html) demonstrates this scenario. To accomplish this task, the user first switches the mode from operational to teach-in by pulling out the safety stop button. Then the user repositions the gripper fingers and issues the 'Okay' command, which autocompletes the remaining wizard steps and resumes the program. Hence, the only interactions between the human and the robot are voice-based and haptic. This enables a safety-certified application to allow minor program changes, which do not affect the results of the risk analysis of the application. This scenario arguably demonstrates the benefits of allowing shop floor employees to

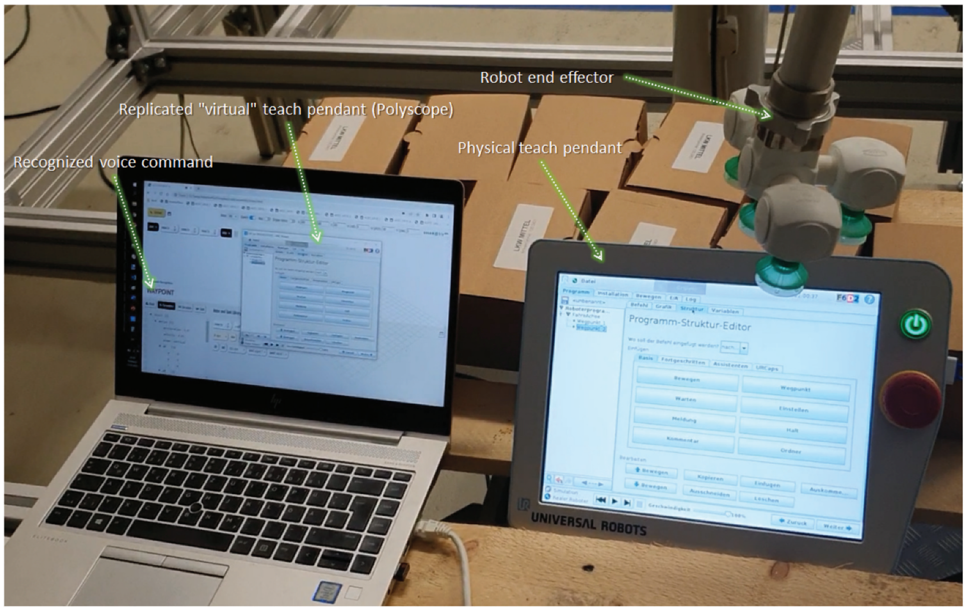


Figure 11. Experimental setup: A UR5 robot, its teach pendant and a laptop connected through Ethernet to the robot's control computer.

change existing robot programs so as to improve their reliability, which is one of the most important production system qualities.

4.2. *UR polyscope*

The validation of robot pose teach-in by voice was conducted in the Aspern Industry 4.0 Pilot Factory in Vienna using a UR5 CB series robot. The voice commands 'waypoint' and 'okay' were used to trigger the creation of waypoints in Polyscope (UR's proprietary programming environment installed on the robot's teach pendant). The robot's HMI was replicated on a computer running the voice recognition in the background using a tool called VNC Viewer. On the robot's control computer, a VNC server was installed. Figure 11 illustrates the experimental setup.

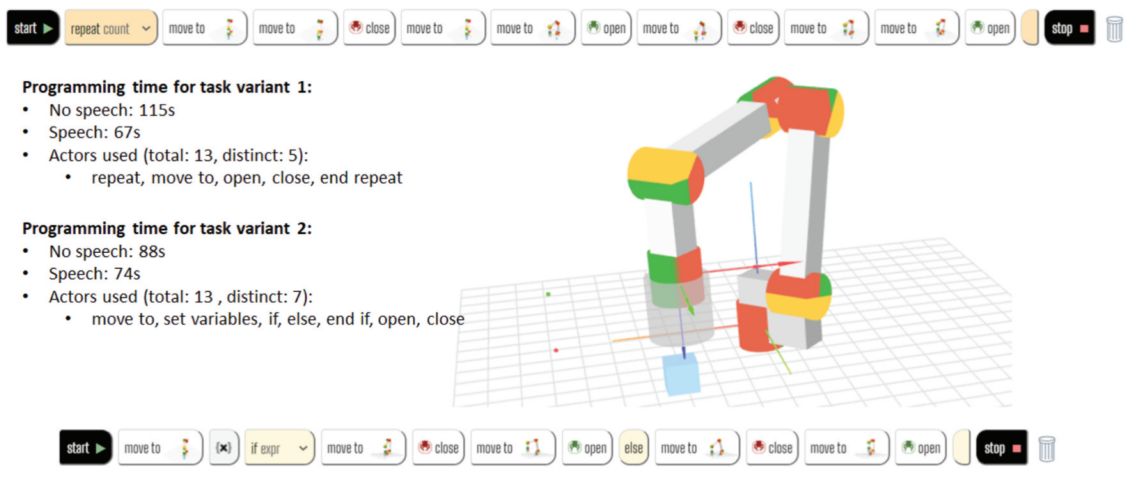
The replicated HMI is displayed on a laptop, which is connected to the robot's control computer through Ethernet. The VNC software instantly replicates the display (from the physical to virtual teach pendant) and the emulated user actions (from the virtual to the physical teach pendant). This allows the GUI automation script, which is running in the background, to click through the menus of Polyscope so as to create a new waypoint and to keep the 'freedrive' button pressed. When the manual positioning of the robot is complete, the user issues to command 'okay' to safe the waypoint. Video number eight on the demo website (https://blockly-desk.comemak.at/demos/code/speech.html) demonstrates this procedure.

The experiment shows that the voice commands allow the user to position the robot with one or both hands since there is no need to perform any manual actions in the robot's HMI – neither on the physical nor on the virtual teach pendant. In the experiment, the user is able to capture the scene using a smart phone while teaching robot poses without having to switch from one device to another.

4.3. *Assembly*

The evaluation of the programming by voice feature that was added to *Assembly* was performed by measuring the time it takes to create a simple pick-and-place task in two variants. The task consisted in picking and placing a cube from one place to another and then back. As shown in Figure 12, in the first variant, the task was implemented using a high number of 'move to' actors, which corresponds to a naïve implementation. In addition, a repeat actor was used to repeat the task for five times. In the second variant, an if-else clause was used to control program execution, depending on the blue cube's position. The experiment was performed by an experienced user of the tool. The tasks are depicted in videos 3–6 linked from the demo web page (https://blockly-desk.comemak.at/demos/code/speech.html).

The measurements show that programming the first task variant by voice is almost twice as fast as programming it using the mouse alone. By contrast, programming the second task variant using voice commands is only 16% faster than programming it using the mouse alone. This contrast is likely due to the varying complexity of the implementations, with the second variant requiring seven different actors as



Programming time for task variant 1:
- No speech: 115s
- Speech: 67s
- Actors used (total: 13, distinct: 5):
  - repeat, move to, open, close, end repeat

Programming time for task variant 2:
- No speech: 88s
- Speech: 74s
- Actors used (total: 13 , distinct: 7):
  - move to, set variables, if, else, end if, open, close

Figure 12. *Assembly*test programs and statistics.

opposed to only five for the first one. In addition, the second variant uses more complex logic, which requires longer reflection times. Another difference between the speech/no speech implementations is that in the former case, the user has to specifically add the 'end repeat'/'end if' actors, whereas in the latter case, these actors are added automatically when adding the 'repeat'/'if' actors by dragging them from the library and dropping them into the task workflow. One limitation of the current implementation of the speech-based approach is that it only allows adding actors at the end of the current sequence. If the user makes a mistake, the 'delete' command can be used to remove the last actor in the sequence. Another limitation is that setting parameter and variable values using the '*set <parameter/ variable name> <value>';* command cannot reliably recognize abbreviated parameter/variable names and complex values, such as strings or expressions. This means that in order to use all features of the *Assembly* tool, the use of a mouse and keyboard cannot be entirely replaced by voice commands. The recent literature on speech-based computer programming (Nowogrodzki 2018), however, provides syntactic solutions for creating variables and setting there values through voice commands. The complexity of the required commands, however, does not justify an implementation for simplified robot programming tools, such as *Assembly*, which specifically target novice programmers. Overall, the evaluation results suggest that programming in *Assembly* by voice significantly increases productivity and improves user experience, since the dragging and dropping of actors can eventually induce physical strains and become annoying. The *Assembly* tool, including the speech-based programming features, can be used online at: https://assembly.comemak.at.

5. Conclusion

This paper introduced a new method and tool for programming cobots by voice. This provides an additional mode of programming cobots, which currently offer multi-modal programming environments based on haptic, mouse, and keyboard interactions. The main benefit of this approach is an increase in the productivity of cobot programmers in scenarios that require a high number of teach-in operations. In addition, a scenario in which shop floor employees can perform small modifications to existing programs without requiring re-certification is explored. Another benefit of the approach is the possibility of communicating cobot programs by demonstration through videos – a feature that may prove useful in training and education contexts.

The first two meta-programming approaches presented in this paper use the same basic principle, namely that of emulating use interaction in the vendor-provided HPI/HMI. This enables the extensions of the respective HMIs by features that the vendor has not foreseen. Moreover, these extensions can go beyond that which is technically feasible by following the plugin approach. While the bookmarklet approach can be applied to extend the interaction modalities of any robot or cobot that provides a web-based HMI, the meta-controller approach can be used with any robot that allows a remote connection to its HMI. The meta-programming approach thus provides an additional means for integrating heterogeneous robotic systems and extending their interaction modalities, which is simpler than the middleware approach or integration via the OPC UA and other protocols. The unique capability of GUI automation applied to robotics consists in allowing extensions of robot HMIs that go beyond what the vendor has imagined and without losing any of the useful features of the vendor-provided HMI. The meta-controller approach eliminates the limitations of the bookmarklet approach, which is only applicable to web-based robot HMIs.

The third approach, which was implemented as an extension to the *Assembly* tool, uses the native capabilities of the browser's document object model (DOM) as well as the jQuery library to endow the tool with speech-based programming capabilities. This approach can be easily adopted by existing commercial web-based robot programming tools, such as Drag & Bot and Blockly-based tools. This paper showed that programming by voice speeds up the programming task considerably, while reducing physical strains and improving the programmer's ability to focus on the task. As opposed to the other two approaches introduced in this paper as well as other approaches from the literature, in *Assembly*, voice commands were successfully used to also instantiate program blocks that go beyond mere robot motions and configurations (e.g. loops and conditions). This aligns the current approach with state-of-the-art speech-based computer programming approaches, such as (Nowogrodzki 2018).

The common aspects between the three approaches include the facts that speech-based programming can be implemented on top of existing tools in a non-intrusive way by automating the user interactions that are necessary to achieve a certain programming goal. In some cases, programming by voice indeed provides a third hand, notably when teaching robot motions either on the screen or through physical manipulation. The results of the evaluation suggest that voice commands are especially useful for replacing repetitive drag and drop tasks, for short-circuiting complex manual operations in various robot programming tools, and for speeding up teach-in tasks. In addition, the user experience is improved by reducing physical strains and annoyance when performing repetitive tasks.

When using the meta-programming approach (be it using a bookmarklet or a meta-controller), some precautions need to be taken, notably concerning safety. The extensions to the original HMI need to be clearly understood by users before interacting with the robot, otherwise they might come as a surprise. From a technical point of view, meta-controller must ensure that interactions with the HMI are mutually exclusive in order to prevent unexpected robot behaviors. Whereas in the case of bookmarklets, it is possible to display warnings and additional information in the original HMI, in the case of meta-controller, such information should be displayed using pop-up messages in the robot's HMI or on a separate screen that is connected to the meta-controller device.

As part of our future work, the programming by voice framework will be evaluated using a crowdsourced user study with 30–40 participants following the methodology introduced by Daria et al., 2021202. In addition, the speech-based programming capabilities in *Assembly* will be further extended to cover all features of the tool. An industrial evaluation in an Austrian engine factory is also planned.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

The work was supported by the Österreichische Forschungsförderungsgesellschaft [871459].

References

Ajaykumar, G., M. Steele, and C.-M. Huang. 2021. "A Survey on End-User Robot Programming." *ACM Computing Surveys* 54 (8): 1–36. doi:10.1145/3466819.

Arnold, S. C., L. Mark, and J. Goldthwaite. 2000. "Programming by Voice, Vocal Programming." Proceedings of the fourth international ACM conference on Assistive technologies. Arlington, Virginia, USA.

Begel, A., and S. L. Graham. 2006. "An Assessment of a Speech-Based Programming Environment." In *Visual LAnguages and Human-Centric Computing (VL/HCC'06)*. Brighton, UK: IEEE.

Beschi, S., D. Fogli, and F. Tampalini. 2019. "CAPIRCI: A Multi-Modal System for Collaborative Robot Programming." In *International Symposium on End User Development*. Cham: Springer.

Bingol, M. C., and O. Aydogmus. 2020. "Performing Predefined Tasks Using the Human–Robot Interaction on Speech Recognition for an Industrial Robot." *Engineering Applications of Artificial Intelligence* 95: 103903. doi:10.1016/j.engappai.2020.103903.

Bugmann, G., and J. N. Pires. 2005. "Robot-By-Voice: Experiments on Commanding an Industrial Robot Using the Human Voice." *Industrial Robot: An International Journal* 32 (6): 505–511. doi:10.1108/01439910510629244.

Daria, P., T. B. Ionescu, and S. Schlund. 2021. "Crowdsourced Evaluation of Robot Programming Environments: Methodology and Application." *Applied Sciences* 11 (22): 10903. doi:10.3390/app112210903.

Drag and Bot GmbH. 2020. "Flexible Produktionsplanung dank einfacher Roboterprogrammierung." *JOT Journal für Oberflächentechnik* 60 (5–6): 32–33. doi:10.1007/s35144-020-0550-2.

Fitzek, F. H., S.-C. Li, S. Speidel, and T. Strufe. 2021. *Tactile Internet with Human-In-The-Loop: New Frontiers of Transdisciplinary Research*. Tactile Internet, 1–19. Cambridge, Massachusetts: Academic Press.

Fraser, N. 2015. "Ten Things We've Learned from Blockly." In 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). Atlanta, GA, USA: IEEE.

Gadre, S. Y., E. Rosen, G. Chien, E. Phillips, S. Tellex, and G. Konidaris. 2019. "End-User Robot Programming Using Mixed Reality." in 2019 International Conference on Robotics and Automation (ICRA). Montreal, Canada: IEEE.

Gkournelos, C., P. Karagiannis, N. Kousi, G. Michalos, S. Koukas, and S. Makris. 2018. "Application of Wearable Devices for Supporting Operators in Human-Robot Cooperative Assembly Tasks." *Procedia CIRP* 76: 177–182. doi:10.1016/j.procir.2018.01.019.

Gustavsson, P., A. Syberfeldt, R. Brewster, and L. Wang. 2017. "Human-Robot Collaboration Demonstrator Combining Speech Recognition and Haptic Control." *Procedia CIRP* 63: 396–401. doi:10.1016/j.procir.2017.03.126.

Halt, L., F. Nagele, P. Tenbrock, and A. Pott. 2018. "Intuitive Constraint-Based Robot Programming for Robotic Assembly Tasks." In 2018 IEEE International Conference on Robotics and Automation (ICRA). Brisbane, Australia: IEEE.

Heimann, O., and J. Krüger. 2018. "Affordance Based Approach to Automatic Program Generation for Industrial Robots in Manufacturing." *Procedia CIRP* 76: 133–137. doi:10.1016/j.procir.2018.01.033.

International Federation of Robotics (IFR). 2019. "Demystifying Collaborative Robotics." Positioning Paper.

Ionescu, T. B. 2020a. "Leveraging Graphical User Interface Automation for Generic Robot Programming." *Robotics* 10 (1): 3. doi:10.3390/robotics10010003.

Ionescu, T. B. 2020b. "Meet Your Personal Cobot, but Don't Touch It Just Yet." In 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN). Virtual Conference: IEEE.

Ionescu should beIonescu, T. B. (2021). Ionescu 2021 should be. *Sensors*, 21(8), 2589. doi:10.3390/s21082589.

Ionescu, T. B., J. Fröhlich, and M. Lachenmayr (2020). "Improving Safeguards and Functionality in Industrial Collaborative Robot HMIs Through GUI Automation." In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). (Vol. 1), 557–564. Vienna, Austria: IEEE.

Ionescu, T. B., and S. Schlund. 2019. "A Participatory Programming Model for Democratizing Cobot Technology in Public and Industrial Fablabs." *Procedia CIRP* 81: 93–98. doi:10.1016/j.procir.2019.03.017.

Kaczmarek, W., J. Panasiuk, S. Borys, and P. Banach. 2020. "Industrial Robot Control by Means of Gestures and Voice Commands in Off-Line and On-Line Mode." *Sensors* 20 (21): 6358. doi:10.3390/s20216358.

Komenda, T., C. Schmidbauer, D. Kames, and S. Schlund. 2021. "Learning to Share-Teaching the Impact of Flexible Task Allocation in Human-Cobot Teams." *SSRN Electronic Journal*. Available at SSRN 3869551. doi:10.2139/ssrn.3869551.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. In 2004 IEEE Symposium on Visual Languages-Human Centric Computing, IEEE, 199–206.

Kumar, A. S., K. Mallikarjuna, A. B. Krishna, P. V. R. D. Prasad, and M. S. V. S. B. Raju 2016." Parametric Studies on Motion Intensity Factors in a Robotic Welding Using Speech Recognition." In 2016 IEEE 6th International Conference on Advanced Computing (IACC) (pp. 415–420). Bhimavaram, India: IEEE.

Lee, J. H., Y. Kim, A. Sang-Gyun, and S.-H. Bae. 2020. "Robot Telekinesis: Application of a Unimanual and Bimanual Object Manipulation Technique to Robot Control." In 2020 IEEE International Conference on Robotics and Automation (ICRA). Paris, France: IEEE.

Leff, A., and J. T. Rayfield 2001. "Web-Application Development Using the Model/View/Controller Design Pattern." In Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference (pp. 118–127). Seattle, WA, USA: IEEE.

Leotta, M., D. Clerissi, F. Ricca, and C. Spadaro 2013. "Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study." In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (pp. 108–113). Luxembourg, Luxembourg: IEEE.

Liu, P. X., A. D. C. Chan, R. Chen, K. Wang, and Y. Zhu. 2005. "Voice Based Robot Control," 2005 IEEE International Conference on Information Acquisition, Hong Kong, China, pp. 5.

Liu, H., T. Fang, T. Zhou, Y. Wang, and L. Wang. 2018. "Deep Learning-Based Multimodal Control Interface for Human-Robot Collaboration." *Procedia CIRP* 72: 3–8. doi:10.1016/j.procir.2018.03.224.

Liu, Y., and Y. Xu. 2019. "Summary of Cloud Robot Research." In 2019 25th International Conference on Automation and Computing (ICAC). Lancaster, UK: IEEE.

Lv, X., M. Zhang, and H. Li 2008. "Robot Control Based on Voice Command." In 2008 IEEE International Conference on Automation and Logistics (pp. 2490–2494). Qingdao, China: IEEE.

Makris, S., P. Tsarouchi, D. Surdilovic, and J. Krüger. 2014. "Intuitive Dual Arm Robot Programming for Assembly Operations." *CIRP Annals* 63 (1): 13–16. doi:10.1016/j.cirp.2014.03.017.

Meng, J., Z. Feng, and T. Xu 2020. "A Method of Fusing Gesture and Speech for Human-Robot Interaction." Proceedings of 2020 the 6th International Conference on Computing and Data Engineering. Sanya, China (pp. 265–269).

Mohs, C., Hurtienne, J. , Kindsmüller, M. C. , Israel, J. H. , Meyer, H. A. & die IUUI Research Group (2006). IUUI – Intuitive Use of User Interfaces: Auf dem Weg zu einer wissenschaftlichen Basis für das Schlagwort "Intuitivit t". *MMI-Interaktiv*, 11: 75–84.

Naumann, A., J. Hurtienne, J. H. Israel, C. Mohs, M. C. Kindsmüller, H. A. Meyer, and S. Hußlein 2007. "Intuitive Use of User Interfaces: Defining a Vague Concept." In International Conference on Engineering Psychology and Cognitive Ergonomics (pp. 128–136). Berlin, Heidelberg: Springer.

Nowogrodzki, A. 2018. "Speaking in Code: How to Program by Voice." *Nature* 559 (7712): 141–143. doi:10.1038/d41586-018-05588-x.

Nuzzi, C., S. Pasinetti, R. Pagani, S. Ghidini, M. Beschi, G. Coffetti, and G. Sansoni. 2021. "MEGURU: A Gesture-Based Robot Program Builder for Meta-Collaborative Workstations." *Robotics and Computer-Integrated Manufacturing* 68: 102085. doi:10.1016/j.rcim.2020.102085.

Pedersen, M. R., Nalpantidis, L., Andersen, R. S., Schou, C., Bøgh, S., Krüger, V., & Madsen, O. (2016). Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, 37, 282–291.

Perzanowski, D., A. C. Schultz, W. Adams, E. Marsh, and M. Bugajska. 2001. "Building a Multimodal Human-Robot Interface." *IEEE intelligent systems* 16 (1): 16–21. doi:10.1109/MIS.2001.1183338.

Piacun, D., Ionescu, T. B., & Schlund, S. (2021). Crowdsourced Evaluation of Robot Programming Environments: Methodology and Application. *Applied Sciences*, 11(22), 10903. doi:10.3390/app112210903.

Pieskä, S., J. Kaarela, and J. Mäkelä. 2018. "Simulation and Programming Experiences of Collaborative Robots for Small-Scale Manufacturing." In *2018 2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*. Cavan, Ireland: IEEE.

Pires, J. N., and A. S. Azar. 2018. "Advances in Robotics for Additive/Hybrid Manufacturing: Robot Control, Speech Interface and Path Planning." *Industrial Robot: An International Journal* 45 (3): 311–327. doi:10.1108/IR-01-2018-0017.

Rogalla, O., M. Ehrenmann, R. Zollner, R. Becher, and R. Dillmann 2002. "Using Gesture and Speech Control for Commanding a Robot Assistant." in Proceedings of IEEE International Workshop on Robot and Human Interactive Communication Berlin, Germany, 454–459.

Rogowski, A. 2013. "Web-Based Remote Voice Control of Robotized Cells." *Robotics and Computer-Integrated Manufacturing* 29 (4): 77–89. doi:10.1016/j.rcim.2012.11.002.

Salem, M., K. Rohlfing, S. Kopp, and F. Joublin 2011. "A Friendly Gesture: Investigating the Effect of Multimodal Robot Behavior in Human-Robot Interaction." In 2011 RO-MAN, Atlanta, GA, USA, (pp. 247–252). IEEE.

Schäfer, P. M., F. Steinmetz, S. Schneyer, T. Bachmann, T. Eiband, F. Samuel Lay, A. Padalkar, C. Sürig, F. Stulp, and K. Nottensteiner. 2021. "Flexible Robotic Assembly Based on Ontological Representation of Tasks, Skills, and Resources." *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* 18 (1): 702–706.

Schmidbauer, C., S. Schlund, T. B. Ionescu, and B. Hader 2020." Adaptive Task Sharing in Human-Robot Interaction in Assembly." In 2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM) (pp. 546–550). Marina Bay Sands, Singapore: IEEE.

Soshi, I., C. J. Paredis, and P. K. Khosla. 2005. "Interactive Multimodal Robot Programming." *The International Journal of Robotics Research* 24 (1): 83–104. doi:10.1177/0278364904049250.

Steinmetz, F., V. Nitsch, and F. Stulp. 2019. "Intuitive Task-Level Programming by Demonstration Through Semantic Skill Recognition." *IEEE Robotics and Automation Letters* 4 (4): 3742–3749. doi:10.1109/LRA.2019.2928782.

Steinmetz, F., A. Wollschläger, and R. Weitschat. 2018. "Razer—a Hri for Visual Task-Level Programming and Intuitive Skill Parameterization." *IEEE Robotics and Automation Letters* 3 (3): 1362–1369. doi:10.1109/LRA.2018.2798300.

Villani, V., F. Pini, F. Leali, C. Secchi, and C. Fantuzzi. 2018. "Survey on Human-Robot Interaction for Robot Programming in Industrial Applications." *Ifac-PapersOnline* 51 (11): 66–71. doi:10.1016/j.ifacol.2018.08.236.

Wang, L., R. Gao, J. Váncza, J. Krüger, X. V. Wang, S. Makris, and G. Chryssolouris. 2019. "Symbiotic Human-Robot Collaborative Assembly." *CIRP Annals* 68 (2): 701–726. doi:10.1016/j.cirp.2019.05.002.

Weintrop, D., A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd, and D. Franklin. 2018. "Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices." Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. Montreal, QC, Canada.

Weintrop, D., D. C. Shepherd, P. Francis, and D. Franklin 2017. "Blockly Goes to Work: Block-Based Programming for Industrial Robots." In 2017 IEEE Blocks and Beyond Workshop (B&B) (pp. 29–36). Raleigh, North Carolina: IEEE.

Wildgrube, F., A. Perzylo, M. Rickert, and A. Knoll. 2019. "Semantic Mates: Intuitive Geometric Constraints for Efficient Assembly Specifications." In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Macau, China: IEEE.

Wilhelm, B., S. Schlund, and C. Vocke. 2017. "Working Life Within a Hybrid World–How Digital Transformation and Agile Structures Affect Human Functions and Increase Quality of Work and Business Performance." In *Advances in Human Factors, Business Management and Leadership. AHFE 2017. Advances in Intelligent Systems and Computing*, edited by J. Kantola, T. Barath, and S. Nazir. Vol. 594. Cham: Springer. 3–10.

Yeh, T., T. H. Chang, and R. C. Miller 2009. "Sikuli: Using GUI Screenshots for Search and Automation." In Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology.(Victoria, BC, Canada: ACM. (pp. 183–192).