

Modular automatic generation of automation software for manufacturing machines

P. Sangregorio, A. L. Cologni and F. Previdi, *Member, IEEE*,

Abstract—Day by day, producers of manufacturing systems are challenged with the need of flexibility and reconfigurability of their products. Being able of rapidly provide various configurations of products in response to customers requests is an important asset in the nowadays competition. Nevertheless, most of the automation software for special machines is built in a monolithic way which does not support rapid reconfigurability to reflect hardware modifications. This paper proposes a modular and reconfigurable approach to software development for the automation to support rapid reconfigurations of the software. The designed strategy consists in a seamless replication of the hardware modularization within the software to allow software generation by assembling pre-built pieces of software. The approach is presented and then applied to a real manufacturing machine.

Index Terms—automation, system engineering, software.

I. INTRODUCTION

In this age of globalization, being able to rapidly respond to customers requests for customization is fundamental to win against competitors. Competition is about speed and costs, which means that being able to respond fast and contain costs is crucial for gain new customers. Until the last decade, manufacturing systems development was based on a design-by-discipline approach as seen in [1]. This kind of approach is strictly sequential, and thus inefficient because each sequence puts constraints on the following ones requiring each team to work only once the previous stages are finished. Since then, many studies have been carried out regarding Systems Engineering and many solutions have been proposed. [2] considered the language SysML as a way to design and represent mechatronic systems, to allow teams with different backgrounds to work on a shared model that covers all the aspects of the product. SysML is a profile of UML dedicated to systems engineering modeling. It is a general-purpose modeling language that supports the analysis, specification,

This research activity has been carried out in the context of the research project *ADAPTIVE, Approccio modulare ed adattativo alla Fabbrica Digitale* funded by Ministero dell'Istruzione, Università e Ricerca and *Touchplant project*, a project funded with the joint support of *Regione Lombardia* and *Fondazione Cariplo*.

P. Sangregorio is with the Control and Automation Laboratory of the department of management, information and production engineering of Università degli Studi di Bergamo, viale Marconi, 4, 24044 Dalmine – Italy (e-mail: paolo.sangregorio@unibg.it).

A. L. Cologni is with the Control and Automation Laboratory of the department of management, information and production engineering of Università degli Studi di Bergamo, viale Marconi, 4, 24044 Dalmine – Italy (e-mail: alberto.cologni@unibg.it).

F. Previdi is with the Control and Automation Laboratory of the department of management, information and production engineering of Università degli Studi di Bergamo, viale Marconi, 4, 24044 Dalmine – Italy (e-mail: fabio.previdi@unibg.it).

design, verification and validation of complex systems such as hardware, software, data, procedures and facilities. Starting from the developed model, [3] proposed a method for automatic generation of automation software. Other solutions have been proposed in the domain of Petri nets, with the purpose of providing a formal model of the software that can be tested with mathematical methods, see [4] and [5]. Once a Petri Net implementation of the automation software is developed, it can be automatically translated into automation software compliant with IEC 61131 standards as reported in [6] and [7]. Another interesting solution for supporting machine customization is the Software Product Lines (SPL) approach. SPL is a set of methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production. Once the feature model is defined (the model that describes the whole possible alternatives and configurations) the software for each possible configuration is developed and when a new machine comes, the features present on that machine are selected and the code automatically generated [8]. However, the SPL approach requires the company to know all the alternatives tree from the very beginning, defining constraints in a predetermined way. This work tries to overcome that limitation by providing a modular methodology for automation software design of small manufacturing systems that can be built by a combination of standard modules and custom ones.

The method described in this paper is based on four steps:

- definition of the product specifications;
- selection of the mechatronic modules;
- development of the control system state machine;
- automatic generation of the HMI and documentation.

The described solution has been tested and verified on a real manufacturing system (an industrial manipulator, see [9] and [10]). This device has characteristics that make it perfect as a test bench. The device is a manipulator that helps workers to lift heavy loads without effort. It is composed by a motor that compensates the load weight using a rope. This basic configuration can be extended with various end-effectors to hang different things, or with additional tools to allow specific movements of the load. The currently available configurations of the device are more than one hundred, and all of them uses a common base of components, set up in different combinations. This makes the device perfect for the application of the proposed approach. Instead of redesigning the software from scratch for each new version of the manipulator, it can be assembled selecting the components previously developed.

This article is structured as follows: in Section II the approach and the field of applicability of the solution is presented, Section III presents an applicative case where the method has been tested with details about tools used and obtained results. At the end, section IV provides a final assessment and future developments.

II. THE APPROACH

Many manufacturing systems are built starting from a common base and a combination of pre-built mechatronic components. When it comes to software instead, this modularity is not always leveraged, causing a waste of time and resources. Many manufacturers in fact, develop the software for newly built machines starting from the software of other similar machines, adapting it to the different hardware. This approach has various risks and is not optimized: first of all, if a bug is discovered in the software of a machine, there is no way to trace which machines have been developed starting from the software that presented that bug. Additionally, this methodology requires each machine software to have a standalone life. Each time an improvement is made in a piece of software of a machine, the only way to replicate the same improvement on all the similar machines is to manually edit the software of all of them. This means that each machine has its own software version, that is not connected in any way to the other similar ones, making the traceability of bugs, issues, improvements almost impossible.

The solution proposed to overcome this issue, aims at separating the automation software in modules to match the physical components that compose the machine. These software modules implement all the control software required for the component to work properly leveraging concepts like information hiding and encapsulation typical of high level languages.

This encapsulation can be achieved by defining software interfaces for communication and data exchange in a similar way of mechanical/electrical interfaces. This allows the internal software of the module to have an independent life as long as it keeps the defined interfaces. A module with the same interfaces of another one can be easily replaced without any change to the rest of the code, just by swapping these modules. This also means that each module can be independently tested, verified and versioned. Each time a bug is identified into a software module, the code can be corrected and all the machines that run that version of the module can be easily found and updated. The modularization helps also during the development phase of the software. Instead of writing the whole automation code, the software can be assembled by composition of the software modules needed, and the only custom code that needs to be developed each time is the working cycle software that coordinates all the modules.

When developing software for manufacturing systems there are other artifacts that are required (like documentation, test checklists, HMI, risk documents). All these artifacts strictly depend on which modules are present on the machine. To make an example, if the machine has components that have blades

there could be a risk of cutting. The structure of a module proposed by the present paper contains additional meta data to support automatic generation of these additional artifacts. These meta-data contains:

- **versioning information** Provides a way for keeping a log of the various versions of the modules allowing tracking of features and issues
- **description and documentation** This data is required for the automatic generation of the documentation. Each module has its documentation in various languages, for supporting the multi-language generation of the user manual
- **interfaces** As stated before, the encapsulation and information hiding are the base concepts for enabling modularity. The interfaces definition specifies how it is possible to read data and invoke actions on this module
- **configurable properties** Each module can have various configuration parameters for changing its behavior (i.e. different configurations of the hardware module that need to be considered in the software)
- **risks associated** Some mechatronic modules can be dangerous for the operators. The risk assessment information need to be linked to the module to be able to build risk documents
- **runtime configurable parameters** Each module can have some parameters that can be changed while operating the machine. These data is linked to the module to be able to automatically generate the configuration HMI

Once modules have been created, they are stored in a versioned database that can be used for composing the software of new machines that contains such modules. Having a versioned database of software modules has many interesting outcomes:

- usage of the latest validated and verified version for each module (instead of relying on copy and paste techniques widely used in the industry, that are error-prone)
- high traceability of issues and bugs in automation software. When an issue is discovered in a software block, all the devices that have the same version can easily be identified and an update of the fixed module can be delivered.
- increase of hierarchical levels in software development: the developer that develops the single software module is not the same that develops the machine working cycle. This allows developers to concentrate on smaller parts of the whole software, increasing the reliability and the quality of the developed code.

The designed process for developing software for new machines is summarized in Figure 1. When an order for a new manufacturing machine comes, and after the hardware configuration of the machine is developed, the modules present in the hardware of the target system are identified. Then, software modules are selected from the library and placed in the workspace. Their interfaces are then connected together in a graphical representation to reproduce the hardware configuration.

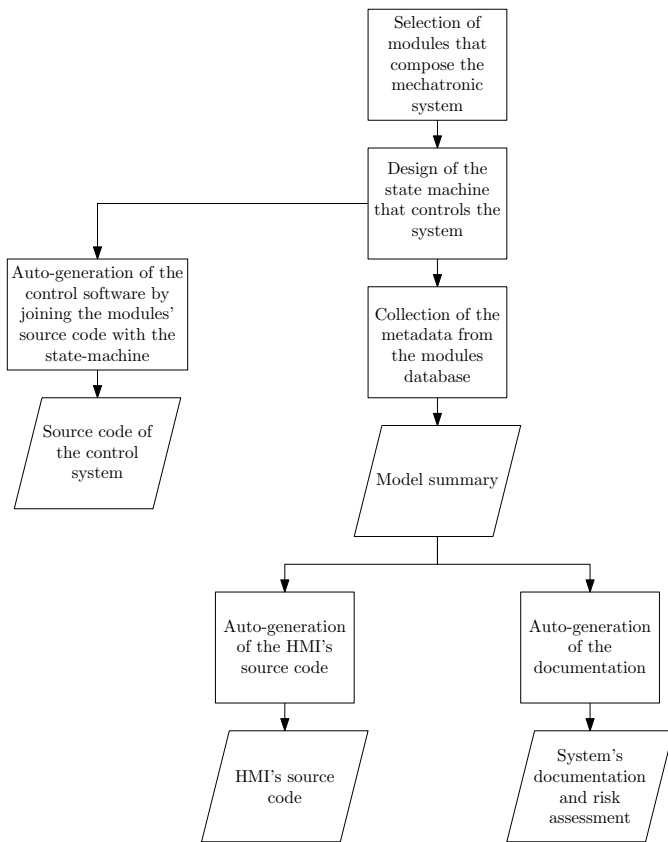


Fig. 1. The proposed approach

After the selection of modules, the automation engineer can concentrate on developing the machine working cycle. This is the most valuable work when realizing a manufacturing system because it is the only part that is really machine dependent. This is one of the most interesting reasons for using the proposed approach: the developer can focus on the most critical part that is different for each machine without worrying about the control software that commands each single mechatronic module. This step is realized using a finite state machine that operates directly on the modules by invoking actions or reading data from them. The transition conditions between one state and another are depending on inputs that comes from software modules (which are fired due to an event that occurs in the mechatronic object) and actions performed by each state are empowered by sending signals to the software modules.

Once all the software has been designed by connecting modules with the state machine and vice-versa, the automatic generation of the automation software can take place. The state machine and the software modules are translated into automation code using code generation tools, and then are joined with a firmware layer necessary for communicating with the hardware.

The used modules' metadata is searched into the modules database for collecting all the information needed for building the documentation, the risk-assessment document and the

HMI source code.

The HMI source code is then generated by composing all the menu items required by each module. Configuration and visualization menus, in fact, are strictly depending on which modules are mounted on the machine, because each module can have parameters that require to be set, or output data that needs to be displayed.

The last step is the generation of the documentation. As said before, each module contains information related to documentation and risk assessment. This allows an automatic generation of these artifacts by composing the pieces of documentation and risks of each module. The development team can then focus only on specific information that need to be inserted in the documentation, without worrying about the modules risks or documentation that are always the same. This way of composing the documentation is useful also in case the behavior of the module is changed, because the related part of the documentation can be updated immediately, allowing a correct versioning of documentation too.

Figure 2 represents the composition of the automation software starting from the database of software modules.

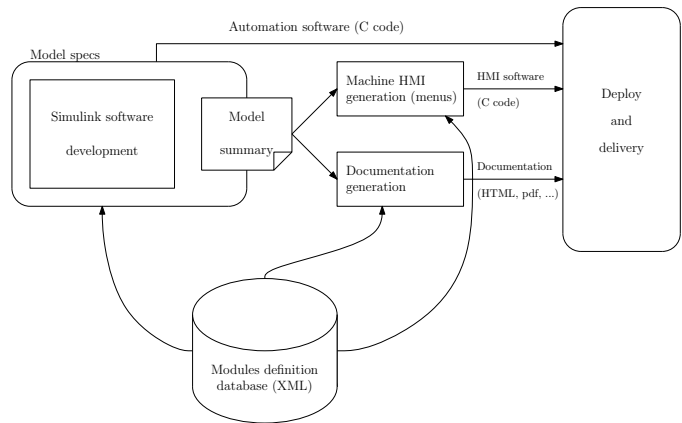


Fig. 2. Composition of the automation software

III. INDUSTRIAL APPLICATION

The presented approach has been applied to an industrial manufacturing system. The selected system is an industrial manipulator, see Figure 3, composed by a vertical body, a motor, an arm and a rope linked to the motor at one end, and to an end effector on the other end. The motor is used to lift the load by compensating its weight. With this mechanism the operator can lift and move the load without any effort because all the weight is compensated by the motor power.

The described part is the standard, common part of all the versions of the device. The part that changes and is available in lots of configurations is the end effector. Depending on the specific application the manipulator is built for, the end effector can have vacuum pumps, grippers, joints, and any other component that is needed for the specific purpose is achieving. This system is suitable for an application of the method described in this paper because the majority of the

automation software is common, regardless of the specific end effector configuration.

Thus, each time a new machine is required, the most of the software can be reused, and the only part that needs to be changed is the end effector related software. In the specific configuration of the manipulator where the method has been applied, the end effector is composed by a gripper, buttons for closing and opening the gripper, a sensorized handle for moving the load and a load balancer that moves back and forth for keeping the center of gravity of the load aligned with the rope.

The first phase is to identify the mechatronic modules that compose the machine:

- *handle*: the handle has sensors for reading the force applied by the operator. It is an input module, the block has only one output interface that provides the read value.
- *load balancer*: the load balancer is basically composed by a gear connected to a rack. When the gear rotates the rack moves back or forth. It is used for keeping the load center of gravity aligned with the rope. This device can be found in two versions, one with two end switches for detecting if the gear reached the end of the rack, and another one with a potentiometer that gives the absolute position of the gear. Depending on the version mounted on the physical device, a parameter of the module has to be set to switch between the two software implementations in the block. The load balancer used in the real case, has two end switches.
- *gripper*: the gripper is used to hang the load. It has two end switches for determining if it is fully open or fully closed. The relative module has an input to receive the command for the gripper, and two outputs to provide the value of the two end switches.
- *input button*: the input buttons are used to opening and closing the gripper. These are just buttons that the operator can press, so they have only one output, the status of the button (pressed / not pressed).
- *load cell*: the load cell is a sensor for reading the load attached to the manipulator. The software module has only one output which gives the value of the attached weight.
- *display*: the display represents the main HMI of the manipulator. The relative software block has only one input, that allows to set the displayed text.
- *lamp*: an output lamp that can be turned on or off. It has only one input to provide the status.
- *motor*: the motor is used for compensate the load hung on the end effector. This module has three inputs: the torque needed for compensating the weight, a maximum speed for lifting, and a maximum speed for dropping (meant for safety reasons).
- *load cell*: the load cell is used for measuring the load of the lifted object. It has basically just one output, the measured weight.

Each module has been described using eXtensible Markup Language (XML) with a scheme defined to contain all the required metadata. The metadata contained is related to ver-

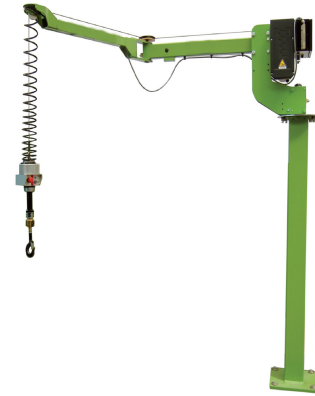


Fig. 3. The industrial manipulator

sioning, interfaces, documentation and parameters. All these information will be used for HMI and documentation generation. After defining the XML metadata, the real development of the module software takes place. The selected environment for the method described in this paper is Matlab Simulink due to its graphical workspace which intrinsically implements the concept of "Block" and due to its ability to automatically generate C code which is supported by the controller of the manufacturing system. Each module is developed into a Simulink block with the same interfaces defined in the XML document. These modules are then stored in a Simulink library.

Once the library is built, the developer can select blocks from the library each time the same mechatronic module is present on the machine to compose the machine software, and concentrate on the development of the machine working cycle.

The machine working cycle is implemented using a Stateflow block for implementing the finite state machine (see Figure 4). This stateflow block uses signals coming from the blocks to trigger transitions between states of the state machine, and provides output signals to the blocks depending on its internal state (see Figure 5). Matlab Stateflow has been selected as tool for representing state machines because it supports static and run-time checks for cyclic problems and state inconsistencies for validation of the developed working cycle.

Each of the blocks represented in Figure 5 internally implements the control logic for the corresponding mechatronic component. Figure 6 represents the internals of the *Balancer* block as an example. The block reads two end switches from the hardware and then, given the position setpoint, commands the motor to move the rack. The output *CurrentPosition* is updated at each movement to provide the current position value as an output of the block.

Within each software module there are some special blocks that are implemented in the main firmware. In this case, the *Input Read* and *Output Write* blocks are used to read and write values on the ECU pins. In the same way, the firmware supports other blocks for other kinds of configuration, like CAN bus blocks. The functionalities provided by those blocks is implemented in the so called *Base firmware* that is the shared basic software that is merged with the machine-specific

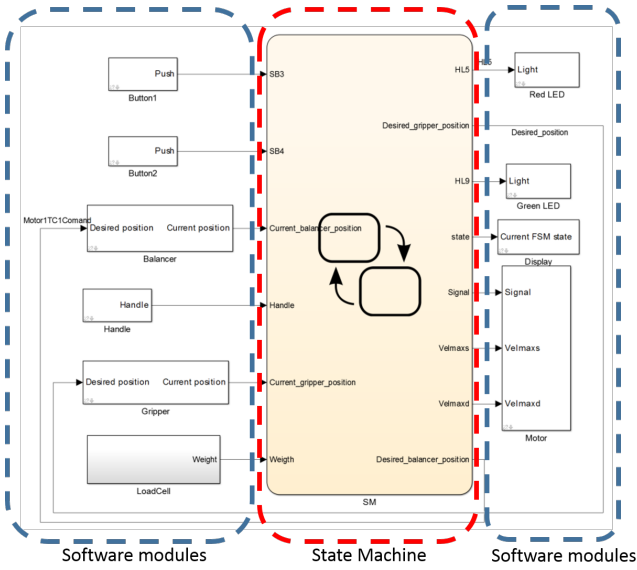


Fig. 4. The connection between software modules and the state machine

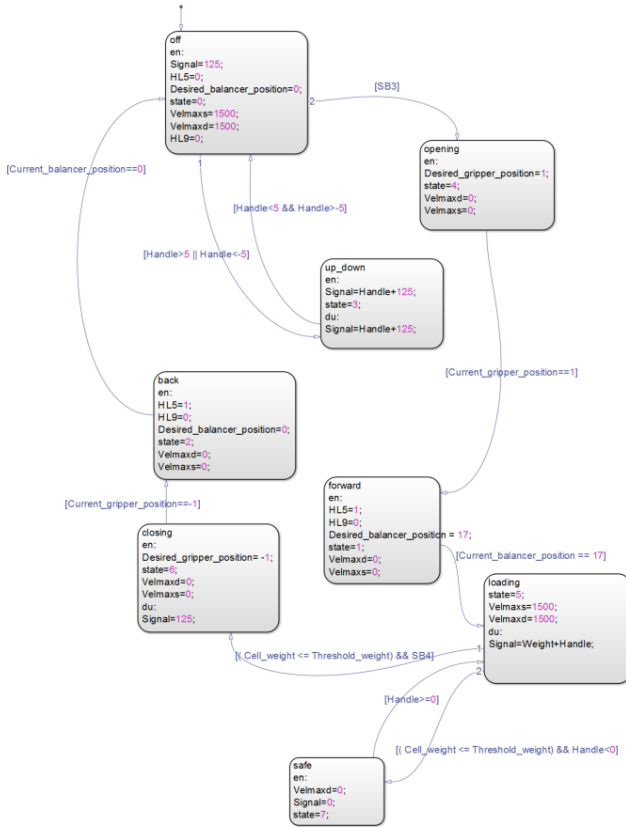


Fig. 5. The content of the finite state machine block

code at deployment. This Base Firmware contains hardware communication functions, the main loop that runs the state machine, the management of the HMI menus and the interrupts for periodic tasks.

Once the software is ready, Simulink Embedded Coder can be triggered to generate a C code implementation of that program. This code is injected into the base firmware at a

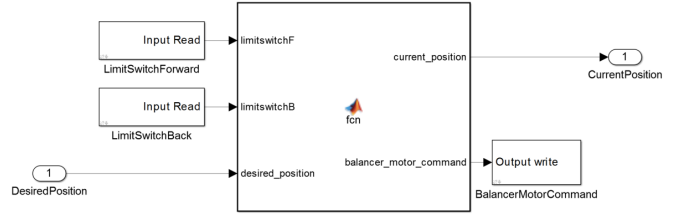


Fig. 6. Internal of the Balancer block

specific point together with the machine HMI program. This HMI program is automatically generated by a tool developed for this application. The tool, collects all the meta-data of the used modules and prepares the HMI menus composing them with all the items that need to be displayed as specified in the XML definition of the modules. The same process is used for generating the software documentation and user manual. Depending on the modules, the corresponding documentation is loaded from the repository and joined in a single document. Thanks to a multilanguage setup of the XML format of the modules, the documentation can be generated in various languages without any effort.

IV. CONCLUSIONS AND FUTURE DEVELOPMENT

In this paper an architecture for the development of small modular and customizable manufacturing systems is presented. The proposed solution is based on prebuilt software blocks that map one-to-one to mechatronic modules, to allow software composition leveraging concepts like modularity and encapsulation. The possibility for the developer to just focus on the working cycle development, without worrying about the implementation of the controllers for each mechatronic part is the most interesting outcome of the solution. This allows a faster time-to-market when developing new machines because the only part that has to be developed is the working cycle, that is the only part that changes between different machines.

Another interesting advantage is about validation: each module follows its own lifecycle and development, which means that is not affected by project deadlines and can be developed in a solid and reliable way. It can be tested and validated in an isolated environment, and thanks to versioning, it is easy to track which version of a module a machine has installed. This allows a fast update of all the machines whenever a bug is discovered on a specific version. Additionally, having a well defined interface, it is straightforward to replace a module with a new implementation of it as long as the interface remains the same.

Future work will consider the possibility to further investigate better architectures for automatically generate the HMI using a specific editor, or a sort of Content Management System (CMS) to allow rapid designing of interfaces with more flexibility. Another interesting research will cover the integration of SysML modeling language with the proposed solution to better modeling the modules and the architecture, taking also in consideration the feature model of the SPL

paradigm, which can be used to formally verify that all the requirements for specific configurations are satisfied.

REFERENCES

- [1] F. Harashima, M. Tomizuka, and T. Fukuda, "Mechatronics-'what is it, why, and how?' an editorial," *IEEE/ASME Transactions on Mechatronics*, vol. 1, no. 1, pp. 1–4, 1996.
- [2] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 3, pp. 1234–1249, 2013.
- [3] N. Papakonstantinou and S. Sierla, "Generating an object oriented iec 61131-3 software product line architecture from sysml," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pp. 1–8, IEEE, 2013.
- [4] G. Music and D. Matko, "Petri net based control of a modular production system," in *Industrial Electronics, 1999. ISIE'99. Proceedings of the IEEE International Symposium on*, vol. 3, pp. 1383–1388, IEEE, 1999.
- [5] M. Chouikha, B. Ober, and E. Schnieder, "Model-based control synthesis for discrete event systems," in *Proc. IASTED Int. Conf. on Modelling and Simulation*, pp. 276–280, Citeseer, 1998.
- [6] G. Mušić, D. Gradišar, and D. Matko, "Iec 61131-3 compliant control code generation from discrete event models," in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pp. 346–351, IEEE, 2005.
- [7] J. Thieme and H.-M. Hanisch, "Model-based generation of modular plc code using iec61131 function blocks," in *Industrial Electronics, 2002. ISIE 2002. Proceedings of the 2002 IEEE International Symposium on*, vol. 1, pp. 199–204, IEEE, 2002.
- [8] N. Papakonstantinou, S. Sierla, and K. Koskinen, "Generating and validating product instances in iec 61131–3 from feature models," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pp. 1–8, IEEE, 2011.
- [9] F. Previdi, F. Fico, D. Belloli, S. Savaresi, I. Pesenti, and C. Spelta, "Virtual reference feedback tuning (vrft) of velocity controller in self-balancing industrial manual manipulators," in *American Control Conference (ACC), 2010*, pp. 1956–1961, IEEE, 2010.
- [10] F. Previdi, F. Fico, S. M. Savaresi, D. Belloli, and I. Pesenti, "Direct design of a velocity controller and load disturbance estimation for a self-balancing industrial manual manipulator," *Mechatronics*, vol. 22, no. 8, pp. 1177–1186, 2012.