A COMPARATIVE STUDY ON EMBEDDED MPC FOR INDUSTRIAL PROCESSES

Adriano Silva Martins Brandão*, Daniel Martins Lima†, Marcus V. Americano da Costa F°‡, Julio Elias Normey-Rico*

*Dep. de Automação e Sistemas, Caixa Postal 476, CEP 88040-900
Universidade Federal de Santa Catarina
Florianópolis, SC, Brasil

†Departamento de Engenharias
Universidade Federal de Santa Catarina
Blumenau, SC, Brasil

‡Departamento de Engenharia Química
Universidade Federal da Bahia
Salvador, BA, Brasil

Emails: adriano.brandao@posgrad.ufsc.br, daniel.lima@ufsc.br, marcus.americano@ufba.br, julio.normey@ufsc.br

Abstract— Predictive control algorithms have great acceptance in the process industry because of its ability to handle multivariable and constrained systems. The presence of constraints greatly increases the computational cost of solving these problems, which has motivated many works in recent years to develop automatic code generation tools for MPC applications. These developments enable the execution of advanced controllers in low-cost embedded systems, which are increasingly applicable to industrial processes due to advances in the development of Internet of Things (IoT) devices. Although several libraries that implement embedded MPC exist, there are not many comparative studies, which makes it difficult to choose among them. The present work aims to compare three of these tools (CVXGEN, FalcOpt and muAO-MPC), to list advantages and disadvantages of some of these tools, to test the performance on an embedded system and to present suggestions of uses and restrictions for applications.

Keywords— Model Predictive Control, embedded hardware, code-generation.

Resumo— Os algoritmos de controle preditivo têm uma grande aceitação na indústria de processos devido à capacidade de lidar com sistemas multivariados e com restrições. A presença de restirções eleva muito o custo computacional da resolução destes problemas, o que motivou muitos trabalhos nos últimos anos a elaborar ferramentas de geração automática de código para aplicações MPC. Esses desenvolvimentos possibilitam a execução de controladores avançados em sistemas embarcados de baixo custo, que são cada vez mais aplicáveis a processos industriais devido aos avanços no desenvolvimento dos dispositivos da Internet das Coisas (IoT). Apesar de várias bibliotecas para a implementação de MPC embarcado existirem, não existem muitos estudos comparativos, o que dificulta a escolha. O presente trabalho visa comparar três desses trabalhos (CVXGEN, FalcOpt e muAO-MPC), citar vantagens e desvantagens de algumas dessas ferramentas, testar num sistema embarcado o desempenho de cada uma e apesentar sugestões de usos e restrições para aplicações.

Palavras-chave— Controle Preditivo, hardware embarcado, geração de código

1 Introduction

Development of IoT (Internet of Things) devices enables the fulfillment of Industry 4.0 standards with distributed connected systems and specialized equipments in the productive process (Brettel et al., 2014). As such, the possibility of using embedded hardware in the process industry, especially in advanced control, is interesting as cheap, efficient and industrial-grade algorithms such as Model Predictive Control (MPC) allows for better performance on resource-constrained industrial processes (Xu et al., 2018). These controllers were developed considering concepts of prediction and calculation of the control signal through the minimization of a certain objective function (Camacho and Bordons, 2002). There are different approaches for improving the efficiency of MPC in the literature such as using interior-point algorithms to solve the optimization (Ding et al., 2016), adapting existing solvers for large-scale problems (Bartlett et al., 2002), using fixed-point arithmetics (Guiggiani et al., 2014) and using Field-Gate Programmable Arays (FPGA) to implement the controller (Xu et al., 2018). This work will focus on code-generation tools that allow the implementation of MPC on embedded hardware.

The number of code-generation tools has been increasing as they can offer execution efficiency and some degree of determinism to the controller. Also, interest on embedded control is rising with the availability of cheap and capable hardware such as the BeagleBone® and Rasberry Pi®, among others. As each tool uses specific formulations of the control problem and offer different capabilities, selection of an appropriate library for implementation of MPC can be hard. The goals of this paper are performing an efficiency comparison of different MPC code-generation libraries using

embedded hardware and providing information to help on the selection of these tools. The tools analyzed by this work are: CVXGEN, a convex quadratic programming solver that provides flexibility on the problem formulation; muAO-MPC, a micro-controller targeted MPC code generation tool that and FalcOpt, an nonlinear predictive control library.

The rest of this paper is divided as follows. In section 2, the predictive controller formulation used in this paper is presented. In section 3, the code-generation libraries and their particularities are described. Simulation results and analysis are in section 4. Finally, the conclusions of this work are in section 5.

1.1 Embedded platform description

The hardware used for the computations presented in this work is a BeagleBone Black Rev-C, a low-power open-source single-board computer designed with open source development in mind. It embeds an 1 GHz ARM processor with 512 MB of DDR3L RAM, 4 GB of eMMC storage and ports such as USB and ethernet. The embedded operating system used on the platform is Debian 9.3, which is provided by the BeagleBoard organization.

2 Model Predicitive Control

One of the most studied predictive control techniques, MPC is a set of advanced control strategies fit for multi-variable constrained processes (García et al., 1989). These techniques solve an optimization problem that minimizes a cost function associated with the predicted behavior of the controlled process (Camacho and Bordons, 2002). Various types of models can be used for the predictions, such as step response and state-space. A generic representation of the problem solved by a linear state-space MPC can be seen in (1) :

$$\min_{U} \sum_{i=1}^{N} \left|\left|X(k+i) - X_{ref}(k+i)\right|\right|_Q^2 + \sum_{i=0}^{N_c-1} \left|\left|U(k+i) - U_{ref}(k+i)\right|\right|_R^2$$

s.t.

$$\begin{aligned} X(k+1) &= AX(k) + BU(k), && k = 0, \ldots, N-1 \\ X(k) &\geq X_{min}, && k = 0, \ldots, N-1 \\ X(k) &\leq X_{max}, && k = 0, \ldots, N-1 \\ U(k) &\geq U_{min}, && k = 0, \ldots, N_c-1 \\ U(k) &\leq U_{max}, && k = 0, \ldots, N_c-1 \end{aligned} \tag{1}$$

Where $X(k)$ and $U(k)$ represent the predicted value of the states and proposed inputs on instant k respectively, the prediction and control horizons are given by N and N_c. The constrains consist of limits to the values of $X(k)$ and $U(k)$ over the prediction horizon and one equality constraint imposing the discrete state-space model dynamics given by the matrices A and B of the model.

The objective function of MPC can be altered in a number of ways to accommodate new control objectives such as economic goals or add robustness to the controller. One of the most used formulations is the one presented in (2), which minimizes the control movements over the horizon instead of the absolute values of the control action.

$$\min_{\Delta U} \sum_{i=1}^{N} \left|\left|X(k+i) - X_{ref}(k+i)\right|\right|_Q^2 + \sum_{i=0}^{N_c-1} \left|\left|\Delta U(k+i)\right|\right|_R^2 \tag{2}$$

This formulation can be used on a tool that only accepts the problem defined in (1) with an augmented prediction model. In this case, a number of states equal to the number of manipulated variables is added to the prediction model. The augmented model is given by the following:

$$\bar{A} = \begin{bmatrix} A & B \\ \mathtt{0} & I \end{bmatrix}, \bar{B} = \begin{bmatrix} B \\ I \end{bmatrix} \tag{3}$$

Where `0` is a matrix of zeros with number of manipulated variables and number of states as number of rows and columns respectively, and I is a square identity matrix and it's size is given by the number of manipulated variables.

The unconstrained linear MPC problem has a global solution that can be computed algebraically, but the use of explicit constraints is one of the main advantages of these controllers as they are critical to most processes. Solving the constrained problem requires the use of a iterative solver, which involves much higher computational costs than those involved on the unconstrained case. This is the main concern when using MPC on embedded systems as computational power available is usually low and the issue that the libraries analyzed on this paper are intended to solve.

3 Code-generation MPC libraries

The use of code-generation tools allows for fast implementation of practical predictive controllers as the low-level aspects of the implementation are mostly handled by the tool and the user can direct its effort on engineering tasks such as design and tuning of the controllers. Using these tools also makes the implementation process less prone to coding errors as the generated code can be assumed to be correct for the given specification of the problem. The code generation libraries create code for solving the MPC optimization problem with various particularities depending on the goals of the library: using static memory allocation to prevent memory leaks; use of only basic

math operations to diminish computation time; taking advantage of sparsity patterns on the matrices to avoid trivial operations, etc.

The use of these tools consists of three phases: Problem specification, on which the MPC problem is defined and usually the numeric values of matrices, constraints and horizons are specified; Code generation, where the tool will use the specified problem to generate code for solving the problem; Code usage, where the generated code is integrated on the desired application, the necessary variables are allocated and the code is compiled and used. All tools analyzed on this work generate C code, which is usually the language used in code-generation as it provides good performance.

The following sections describe the libraries selected for this comparative study. They were chosen considering diversity of scopes (generic purpose, linear and nonlinear MPC), free availably and existence of documentation on how to use the generated code.

3.1 muAO-MPC

Developed in the Laboratory for Systems Theory and Automatic Control of the Otto-von-Guericke University from Germany, this package allows for design, simulation and code generation of MPC controllers with a fixed structure, similar to the one presented in (1) using a combination of augmented Lagrangian with fast gradient methods to solve the optimization. The generated code is fully compatible with the ISO C89/C90 standard and is platform independent (Zometa et al., 2013). It was developed targeting embedded hardware and its controllers have a small footprint in code size and memory usage. The software is free and released under the terms of the three-clause BSD License.

When using this tool the user needs to follow the pattern described in the beginning of the section. To define the model, horizons and constraints, a script must be prepared providing these informations to specifically named variables described on the documentation. The model can be given in discrete or continuous form, the inequality constraints can contain linear combinations of inputs and controls and the control horizon is always equal to the prediction horizon. The code generation is performed using one simple command and this process also generates a makefile for compiling the C code. To use the generated code in an application, it is necessary to include the headers provided on the *mpc.h* header file, allocate a specific structure and set the number of iterations of the algorithm. The controller is called using the *mpc_ctl_solve_problem* function that needs the measurements and the previously allocated structure as inputs, and all matrices are handled in column-major form.

3.2 FalcOpt

This library consists of an implementation of an Projected Gradient Descent method for solving convex nonlinear Model Predictive Control problems and was developed at the Automatic Control Laboratory, ETH Zurich, Switzerland. This algorithm does not support constraints on the states over the prediction horizon, only allowing for quadratic terminal constraint on the states. The manipulated variables can have box constraints and the process model can be nonlinear, although this paper will only consider linear models. The objective function is also customizable, with the only limitation being that it must be a convex function.

Different from the other libraries used in the paper, FalcOpt requires the use of Matlab at the problem definition phase as the model and controller parameters are defined in a Matlab script. The generated code has a Mex interface that allows for use of the controller in Matlab and can be used as a guideline for allocating the necessary memory if used on embedded context. The library requires allocation of numerous data structures for it to be used and does not provide functions for performing the allocations, as muAO-MPC does. The generated code consists of two files that are easy to compile, but it does not provide the necessary header to include the controller in a C program. The header provides an interface to the controller code so the user must build a header based on the necessary functions of the generated code in order to use this library. The controller is called using the function *proposed_algorithm* and the numerous allocated variables must be passed to it at every call.

3.3 CVXGEN

CVXGEN is a web based code generation tool for solving convex quadratic optimization problems (Mattingley and Boyd, 2012) and was developed at Stanford University. It uses a high-level language for stating the optimization problem and generates self-contained C code that implements a tailored solver. The high-level language allows for easy definition of the prediction model, auxiliary variables, constraints and objective function. At the problem definition phase no numeric values must be provided for the parameters, instead only the sizes are defined off-line, and the inputs passed to the generated code when used. CVXGEN is not specific for MPC, therefore requires the user to assemble the whole optimization problem, which is not required by the other tools presented in this paper. This aspect of the tool makes it more flexible to changes on the formulation of MPC, but the code generation process does not work well for systems with a high number of constraints or long horizons as the permutations performed by

the web code generator become increasingly complex. The web interface warns the user if the problem is too complex and may not be possible to generate the code.

Different from the other libraries presented in this work, CVXGEN requires four general steps for using its code: Problem definition, code generation, parametrization and call to the controller. The problem definition is done on an web interface, in which the sizes of parameters such as the model matrices and number of states are defined, then the user must also define the objective function and its constraints. The code generation is also performed on the web interface and can take a few minutes depending on the problem size. The generated code consists of numerous files that can be compiled using the provided makefile. To call the controller, it is necessary to include some headers and declare some specific variables from the solver. Then, the values of weight matrices, model matrices, box constraints and other parameters must be defined. The controller the can be called using the *solve* function, which takes no inputs as the parameters of the solver are all defined as global variables.

3.4 Qualitative analysis

The problem statement on each tool is done on different ways: CVXGEN provides easy formulation of the problem with a flexible high-level interface but requiring most of the parameters to be given at runtime; muAO requires the use of python to provide the problem parameters; FalcOpt uses Matlab, allowing easy matrix manipulations and formulation of the problem. Both muAO and FalcOpt natively provide the use of future reference values on the controller, but need all the references to be passed for all of the prediction horizon even when the use of a constant reference is intended. However, the flexibility of CVXGEN formulations enables the use of future and constant references, without needing references to be passed for the whole prediction horizon when fixed references are intended. The limited constraints of FalcOpt makes the use of this library unpractical on various cases, but the possibility of using fast nonlinear control is an interesting unique feature among the analyzed tools. The generated code form CVXGEN and muAO creates numerous files, but the provided makefiles and headers aid when compiling the code. FalcOpt does not provide a header, but it is simple to create and compiling the generated files are simple to compile.

4 Simulations

The goal of the simulations is to evaluate de performance of the controllers considering the process response and time taken to compute the control action, as the latter is an indicative of implementation efficiency. As such, simulations were implemented on the BeagleBone Black and repeated for 1000 times at each simulated case and for each studied library. The simulation cases are described as the following:

- Case 1: The objective function presented in (1) is used and the states have initial conditions different from the initial reference and references constant over the prediction horizon;
- Case 2: The objective function presented in (1) is used and the states have initial conditions different from the initial reference and future references are passed to the controllers;
- Case 3: The objective function presented in (2) is used, the states have initial conditions different from the initial reference and references constant over the prediction horizon.

All the simulations were performed for 100 iterations with reference changes at time $k = 40$. Off-line and on-line indicators were used as evaluation criteria: size of the generated code and the time needed by the controllers to perform the calculations. All the controllers were tuned with the same parameters: prediction horizon $N = 20$, control horizon $N_c = 3$ when possible to use, and weighting matrices Q and R equal to identity matrices of appropriate size.

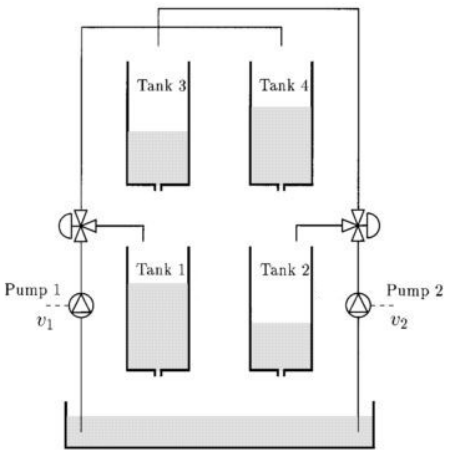

Figure 1: Schematic diagram of the process showing the two pumps (manipulated variables) and the tank levels. Adapted from (Johansson, 2000).

4.1 The process model

The quadruple tank process is a process designed for evaluation of multi-variable control techniques. One of most interesting features is the presence of an adjustable multi-variable zero, allowing for diversity of behaviors on a simple laboratory setup.

Table 1: Summary of characteristics of the libraries.

	Obj. Func.	*Constraints*	*Future reference*	*Control horizon*	*Code complexity*
CVXGEN	Flexible	Flexible	Dependent on formulation	Adjustable	Multiple files, provides Makefile and headers
FalcOpt	Fixed	Only on controls	Yes	Equal to prediction horizon	Two files, does not provide headers
muAO	Fixed	States, controls and combiantions	Yes	Equal to prediction horizon	Multiple files, provides Makefile and headers

The process consists of four interconnected tanks with two pumps that can be used to regulate the inlet flow for the tanks, as shown in Figure 1. The nonlinear model for this process is presented by (Johansson, 2000), but this work will use a discrete linearized state-space model with sampling time of $T_s = 15s$. The matrices of the model, operating point (X_0, U_0) and the operational limits that were used are as follows:

$$A = \begin{bmatrix} 0.7887 & 0 & 0.2820 & 0 \\ 0 & 0.8486 & 0 & 0.2155 \\ 0 & 0 & 0.6808 & 0 \\ 0 & 0 & 0 & 0.7654 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.6439 & 0.1823 \\ 0.0973 & 0.4836 \\ 0 & 0.9657 \\ 0.7362 & 0 \end{bmatrix}$$

$$X_{min} = \begin{bmatrix} -11.3400 & -11.7000 & -4.3200 & -4.4100 \end{bmatrix}^T$$
$$X_{max} = \begin{bmatrix} 12.6000 & 13.0000 & 4.8000 & 4.9000 \end{bmatrix}^T$$
$$U_{min} = \begin{bmatrix} -2.8350 & -2.8350 \end{bmatrix}^T$$
$$U_{max} = \begin{bmatrix} 3.1500 & 3.1500 \end{bmatrix}^T$$
$$X_0 = \begin{bmatrix} 12.6 & 13.0 & 4.80 & 4.90 \end{bmatrix}^T$$
$$U_0 = \begin{bmatrix} 3.1500 & 3.1500 \end{bmatrix}^T$$

4.2 Results

The data from the simulations is presented in three types of graphics: histograms showing the distribution of the run times of the controllers obtained from the 1000 repetitions of each case; a time trend showing the behavior of the run times of the controllers in one simulation over the simulation time and; graphics showing the temporal behavior of the average level of tank 1 (X_1) and the two manipulated variables (U_1 and U_2) over the simulation time.

Observing the histogram and time trend presented in Figure 2, it is evident that FalcOpt presents a less consistent behavior when compared with the other libraries. At the time of the reference change, computing time for FalcOpt increases swiftly for a brief period of time while CVXGEN and muAO do not show this behavior. It is also noticeable that FalcOpt can compute the control action on a time much smaller than the other libraries despite being an nonlinear solver, which may be due to the lack of state constrains in this library. Analyzing the data presented on 2 the above analysis is reaffirmed as the standard deviation of the FalcOpt data is more than ten times the standard deviation of CVXGEN and muAO, although having an minimum time seven times smaller. Considering the closed-loop response, the behavior of the states and manipulated variables are very similar when comparing FalcOpt and muAO, but CVXGEN presents considerable oscillation on the manipulated variables.

Table 2: Computation time statistics for case 1 in seconds.

	Mean	*Std.dev.*	*Min*	*Max*
CVXGEN	7.60e-3	2.0e-4	7.6e-3	1.2e-2
FalcOpt	0.16e-3	26.0e-4	0.1e-3	1.5e-2
muAO	11.9e-3	0.6e-4	11.8e-3	1.4e-2

The analysis of Case 2 results are presented in Figure 4, where it can be seen that FalcOpt increases the computation time as soon as the change on the reference is perceived on the prediction horizon and remains high until the state is close to the reference. This behavior is not present on the other libraries. CVXGEN and muAO show similar performance to Case 1, although with higher average computation times, as seen in Table 3. The behavior of states and inputs is very similar among the controllers, with CVXGEN not presenting the oscillatory behavior of Case 1.

Table 3: Computation time statistics for case 2 in seconds.

	Mean	*Std.dev.*	*Min*	*Max*
CVXGEN	8.6e-3	0.2e-3	8.3e-3	1.3e-2
FalcOpt	3.2e-3	4.6e-3	0.1e-3	1.9e-2
muAO	12.4e-3	0.7e-4	12.3e-3	1.4e-2

On the last simulation case, the analysis of Figure 6 shows that the computation time of

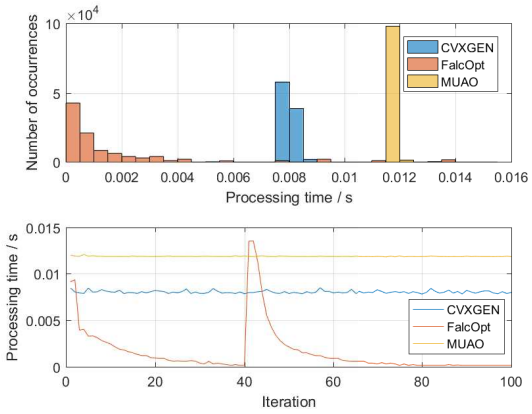

Figure 2: Execution time histogram and behavior over the simulation of case 1.

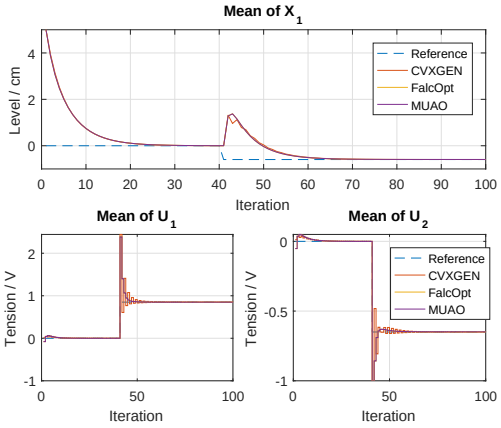

Figure 3: States and controls behavior of case 1.

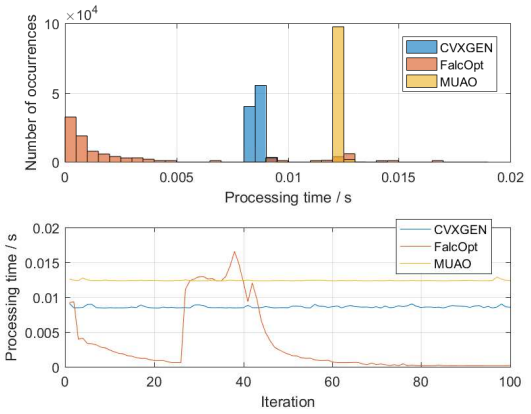

Figure 4: Execution time histogram and behavior over the simulation of case 2.

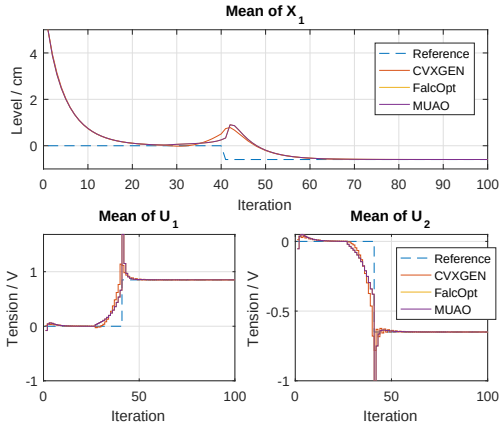

Figure 5: States and controls behavior of case 2.

CVXGEN is affected by the change in reference, but in a different manner as the one presented by FalcOpt on Case 2, the computing time of CVXGEN stabilizes very fast when compared with the latter. Presenting the same behavior of Cases 1 and 2, muAO does not change its computation time by much. In Table 4 muAO is shown to have very consistent times, with the smallest standard deviation. FalcOpt was not able to follow the reference and computation times observed on this case were on average more than ten times greater than the observed on other libraries, as seen on Table 4. As such, Figure 6 presents the time results for CVXGEN and muAO only in order to facilitate visualization.

Table 4: Computation time statistics for case 3 in seconds.

	Mean	*Std.dev.*	*Min*	*Max*
CVXGEN	1.7e-2	2.7e-3	1.5e-2	2.6e-2
FalcOpt	6.4e-1	4.3e-1	0.9e-3	9.6e-1
muAO	1.7e-2	0.9e-4	1.6e-2	1.9e-2

Table 5 shows the size of the generated code for each library. As FalcOpt and muAO by nature allow for future references, the size of the generated code is the same on Case 1 and 2. It seems that the formulation affects more the size of CVXGEN code than the other libraries as it shows the biggest and more variable code in size. FalcOpt code is considerably smaller than the other libraries, being more than ten times smaller on all cases. It is important to notice that all libraries in all cases were able to produce an output in times considerably below the sampling time of the controller.

Table 5: Size of the generated code.

	Case 1	*Case 2*	*Case 3*
CVXGEN	2.18 MB	2.22 MB	1.68 MB
FalcOpt	100 KB	100 KB	104 KB
muAO	1.24 MB	1.24 MB	1.29 MB

5 Concluding remarks

A performance comparison is presented in this paper, showing differences on computing time, code size and temporal behavior of three different code generation libraries for MPC controllers. It is shown that muAO has a very consistent execution time when compared to the other libraries, signal-

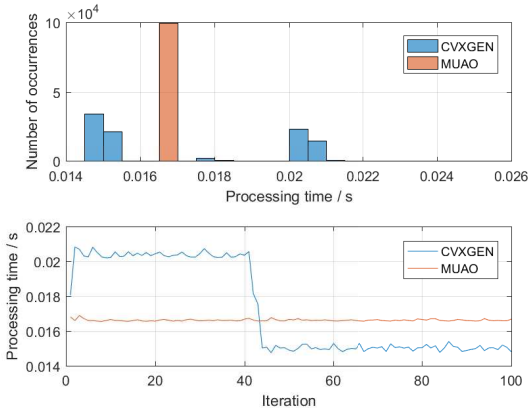

Figure 6: Execution time histogram and behavior over the simulation of case 3.

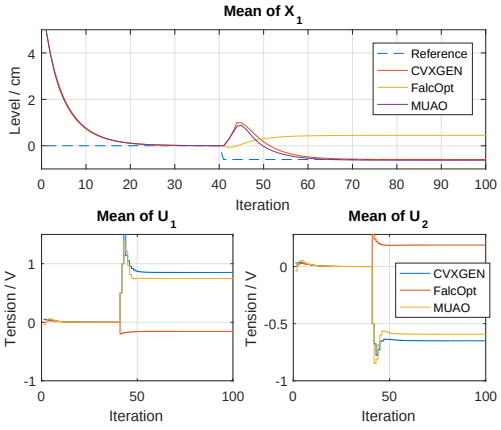

Figure 7: States and controls behavior of case 3.

ing that it is a good option for implementations in which the computing time of the controller is similar to the sampling time, as it provides a smaller possibility of the controller not being able to compute the desired input on time. The integration of this tool on existing code can be performed without much hassle as it provides the necessary tools for including and compiling the generated code. Although having a fixed structure for the MPC problem, the use of python to define the problem allows for easy construction of augmented systems for achieving different formulations. CVXGEN is shown to have undesirable behavior of the manipulated variables with certain formulation, but presented a consistent runtime. As this library allows for changes on the problem online, it certainly can be used with adaptative algorithms, and as it has a very flexible interface for problem formulation, it is an interesting choice when variations of the standard MPC problem are intended. The use of the generated code is simple and the compilation process is similar to the used with muAO. Having very inconsistent computation times, the use of FalcOpt is not advised when the controller computing time is similar to the sampling time, as the computation time can increase to values grater than the sampling time. Although the inconsistent performance, this library is an interesting choice for simple nonlinear MPC problems as it allows for the use of nonlinear models, but the lack of state constraints can limit the usability of this controller on real processes. The use of Matlab to formulate the control problem, although providing easy manipulation of the matrices involved, can limit the use of FalcOpt as Matlab isn't free software. Regardless of the tools chosen by the user, considering the results obtained from this work, all the analyzed libraries are capable of working with small sampling periods, even considering limited a limited computing system. With the above exposed, it is expected that users are able to select the adequate MPC library with more ease.

Acknowledgments

The authors recognize the support provided by the Federal University of Santa Catarina (UFSC), Brazilian National Council for Scientific and Technological Development (CNPq), projects 305785/2015-0 and 311024/2015-7; and by the Human Resources Training Program PRH34-ANP and PFRH34-Petrobras.

References

Bartlett, R. A., Biegler, L. T., Backstrom, J. and Gopal, V. (2002). Quadratic programming algorithms for large-scale model predictive control, *Journal of Process Control* **12**(7): 775–795.

Brettel, M., Friederichsen, N., Keller, M. and Rosenberg, M. (2014). How-Virtualization-Decentralization-and-Network-Building-Change-the-Manufacturing-Landscape–An-Industry-40-Perspective, **8**(1): 37–44.

Camacho, E. F. and Bordons, C. (2002). *Model predictive control*, Vol. 1, 2 edn, Springer.

Ding, Y., Xu, Z., Zhao, J., Wang, K. and Shao, Z. (2016). Embedded MPC Controller Based on Interior-Point Method with Convergence Depth Control, *Asian Journal of Control* **18**(6): 2064–2077.

García, C. E. S. D. C., Prett, D. M. S. D. C. and Morari, M. (1989). Model predictive control: Theory and practice-A survey, *Automatica* **25**(3): 335–348.

Guiggiani, A., Patrinos, P. and Bemporad, A. (2014). Fixed-point implementation of a proximal newton method for embedded model predictive control, *IFAC Proceedings Volumes (IFAC-PapersOnline)* **19**(2012): 2921–2926.

Johansson, K. (2000). The quadruple-tank process: a multivariable laboratory process with\nan adjustable zero, *IEEE Transactions on Control Systems Technology* **8**(3): 456–465.

Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization, *Optimization and Engineering* **13**(1): 1–27.

Xu, Y., Li, D., Xi, Y., Lan, J. and Jiang, T. (2018). Improved Predictive Controller on FPGA by Hardware Matrix Inversion, *IEEE Transactions on Industrial Electronics* **0046**(c): 1–1.

Zometa, P., Kogel, M. and Findeisen, R. (2013). μAO-MPC: A free code generation tool for embedded real-time linear model predictive control, *American Control Conference (ACC), 2013* (1): 5320–5325.