INTERPOLATED MODEL PREDICTIVE CONTROL: HAVING YOUR CAKE AND EATING IT TOO

J. Currie[1], D.I. Wilson[2]

[1]Electrical & Electronic Engineering
AUT University
Auckland, New Zealand
jonathan.currie@aut.ac.nz

[2]Industrial Information & Control Centre
AUT University
Auckland, New Zealand

ABSTRACT

Model Predictive Control or MPC is an advanced, high-performing constraint-aware controller that requires considerable computational demands of the internal constrained optimizer. Consequently MPC has historically been ill-suited for embedded controllers designed to tackle high-speed applications. This paper explores the options of developing a low-cost lightweight MPC controller destined for microcontrollers for modest sized high-speed applications. To obtain the necessary sampling rates, our algorithm runs the full MPC controller calculation to generate the present, and future, control moves only once every 3rd or so sample, thereby reducing the calculation load by one third. In between these full updates, we run a local PI(D) controller using not the external setpoint, but the MPCs model predictions as the setpoint for the controller. This is used in an interpolated fashion as opposed to a pure cascade. We show that we retain almost all the advantages of the MPC (honouring constraints, handling pathological plants, easy to tune etc), and save considerable computation such that it is suitable for implementation in an embedded system.

INTRODUCTION

Model Predictive Control, or MPC, is a one of few examples of advanced process control that is highly regarded by industrial practitioners, Garcia et al. (1989), Qin and Badgwell (2003) and Wilson and Young (2006). While the numbers of MPC installations will never match the ubiquitous PI/PID controller, they do exhibit some clear advantages such as elegant constraint handling, intuitive tuning, and the ability to optimally handle non-square systems.

However the implementation of an MPC application requires considerably more cost and engineering effort in the creation of the model, the development of the optimisation algorithm, and the installation of the controller and support software on suitable hardware. Traditionally this has restricted MPC applications to those processes which were economically important to the bottom line which typically meant large, key multi-variable processing units. Furthermore the optimality characteristic of the controller, while desirable (not the least because it is free from obscure tuning parameters), comes at a computational cost. These reasons may also explain the paucity of MPC

applications outside the process control community as also pointed out recently in Dua et al. (2008).

Recently however there has been activity in porting the MPC algorithm to control problems outside the traditional process control field into areas such as gas and steam turbine control, micro-reactors, mechatronics and biotronics. In all these fields the plants to be controlled are fast-acting requiring kilo-Hertz sampling rates, and in some applications it is desirable to mount the controller remotely, perhaps with power supply limitations. One way to utilise the benefits of an MPC controller in these types of applications is to run the MPC controller in a modern embedded micro-controller. Currently these microcontrollers are small, exhibit low power requirements, are fast (200MHz to 1GHz) and are single-precision-floating point.

This paper describes how one can implement an MPC on a modern embedded controller at high sampling frequencies. If the controller has a single core, the MPC runs as normal, but when the sampling generates an interrupt and the MPC is still computing the current optimal trajectory, the algorithm subroutines to the essentially instantaneous PID controller and exports that as a temporary measure. When the MPC algorithm does eventually finish, typically after around 2 or 3 samples, the optimal input is used, and the cycle repeated.

THE JMPC TOOLBOX

For this development we are using the jMPC Toolbox which is a Model Predictive Control Toolbox we have developed around the Matlab environment, Currie (2011). Simple functionality such as inequality constraints can be placed on inputs and outputs, as well as standard MPC options like state estimation, control move blocking and soft constraints, all of which can all be added with one line of code. Notable advanced functionality includes viewing real-time MPC control with the supplied graphical interface, interfacing to external hardware using the Simulink jMPC block and an object designed specifically for nonlinear simulations.

MPC Implementation

Our implementation of an MPC controller transforms a Linear Time Invariant (LTI) discrete state-space model with possible linear constraints into a quadratic optimization program. Essentially the MPC algorithm delivers a future sequence of control moves, $\mathbf{\Delta u}$ over the immediate future control horizon, N_c, such that the cost function

$$j = \sum_{j=1}^{N_p} \mathbf{q}_j \|\hat{\mathbf{y}}_{k+j|k} - \mathbf{y}^*_{k+j|k}\|^2 + \sum_{j=1}^{N_c} \mathbf{r}_j \|\mathbf{\Delta u}_{k+j|k}\|^2$$

is minimised. The vectors $\hat{\mathbf{y}}$, $\mathbf{y}^*$. are the estimated future output of the plant, and future setpoints, and $\mathbf{q}$ and $\mathbf{r}$ are weighting factors. The prediction, N_p, and control, N_c, horizons are important tuning parameters. The objective function is constrained by linear plant dynamics and possible linear output, input, and input rate constraints. With some algebraic manipulation (see Wang (2009), Rossiter (2004) for the details), the objective function and constraints can be re-written as

$$\min_{\mathbf{\Delta u}} j = \frac{1}{2}\mathbf{\Delta u}^T\mathbf{H}\mathbf{\Delta u} + \mathbf{f}^T\mathbf{\Delta u}$$
$$\text{subject to:} \qquad \mathbf{A}\mathbf{\Delta u} \leq \mathbf{b}$$

which is a standard Quadratic Program (QP) to be solved each sample time using a standard solver such as Wright's Infeasible Interior Point (IIP) Wright (1997). The following subsection will introduce one of our benchmark problems for Interpolated MPC which will be solved in the following example using standard MPC in the jMPC Toolbox.

Nonlinear CSTR

A challenging example is the nonlinear Continuously Stirred Tank Reactor (CSTR) model described in Henson and Seborg (1997). The CSTR exhibits highly nonlinear behaviour and presents a difficult problem for most process control strategies. The CSTR model used is for a single tank with a single reaction (A → B), with a complete material and energy balance, and is described by the following equations

$$\begin{aligned}
\sigma &= k_0 e^{\frac{-E}{RT_r}} C_a \\
\frac{\mathrm{d}\,C_a}{\mathrm{d}\,t} &= \frac{q}{V}\left(C_{af} - C_a\right) - \sigma \\
\frac{\mathrm{d}\,T_r}{\mathrm{d}\,t} &= \frac{q}{V}\left(T_f - T_r\right) + \frac{H}{C_p\rho}\sigma + \frac{UA}{C_p\rho V}\left(T_c - T_r\right)
\end{aligned}$$

with values of parameters as specified in the original reference. With regard to our control problem, the system has one input, two measured disturbances, and two outputs. The system input is the jacket cooling fluid temperature (T_c), the measured disturbances are the feed concentration (C_{af}) and feed temperature (T_f), while the outputs are the reactor concentration (C_a) and reactor temperature (T_r). The objective of the problem is to control the reactor temperature by adjusting the cooling fluid temperature.

As will be demonstrated in Figure 5, a suitably tuned linear MPC can adequately respond to large setpoint and disturbance changes. Furthermore, only a minimum amount of code is required to setup and execute a constrained MPC controller for a non-trivial scenario like the CSTR.

PARALLELIZING MPC

As indicated in the introduction, typical current MPC implementations are restricted to processes with long time constants due to the computational complexity of MPC. Previously, we have profiled a typical MPC application to identify the proportion of computational work allocated to each task of solving the MPC control law, Currie and Wilson (2010). Our aim was to work out the main areas into which we could divide the program, and to see how long each one took to execute.

The results of the timing breakdown the non-minimum phase system presented in the Case Studies section are shown in Figure 1. What is evident, and well established in the literature, is that solving the QP optimization problem can take two or more times as

long as all the other sections combined. Therefore to get a significant performance speed up in the control algorithm we need to focus on how the QP can be solved faster.

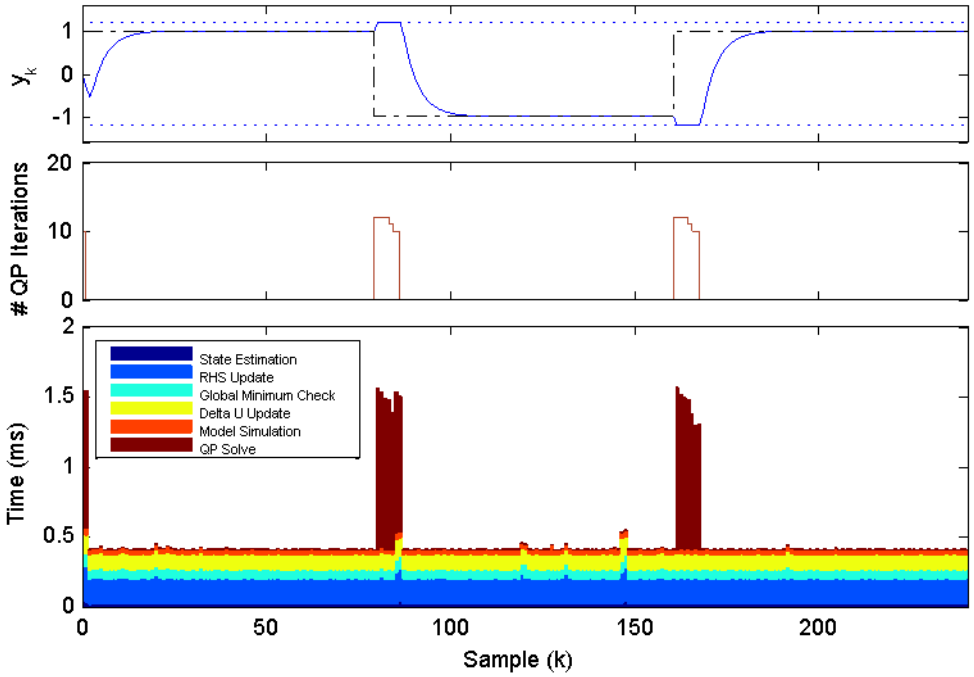

Figure 1: Timing breakdown of the non-minimum phase simulation.

Hastening the QP

Several techniques have already been suggested by other researchers, in order to obtain faster QP convergence and solving for the MPC algorithm. These include reducing the number of decision variables by introducing control move blocking, Cagienard et al. (2007), parametric representations of MPC with pre-computed solutions, Tondel et al. (2003) as well as using a linear cost function as proposed in Rossiter (2004). QP algorithms have also been optimized to reduce solution time using techniques such as warm starting, step length scaling, pre-factorization and other numerical techniques.

However it is to be noted that parallelization is not yet an established technique, and for good reason. All standard optimization algorithms are faced with successive iterations which rely on information from the previous iteration step. This is a major roadblock for parallelizing any algorithm and one that has continued to impede spreading QP optimization workload across the now common multiple cores (with the exception of the linear algebra).

Therefore since we cannot easily parallelize the optimization algorithm, one should ask if it is possible to avoid running it completely. Exploiting the global minimum, as illustrated in Figure 1 bypasses the associated computation time of solving a QP. However while this can increase overall efficiency, it does not increase our maximum sampling frequency, which is governed by the sample with the longest computation time. In order to increase our sampling frequency we need to either reduce the computation time, or spread it out over more than one sample and attempt to approximate the control of MPC in between.

INTERPOLATED MPC

Given the complexity of further speeding up the optimization solver, we decided to see if we could spread the calculation over more than one sample whilst still retaining the optimality of MPC. Therefore we propose running the MPC calculation once every *n* samples, thus allowing a longer calculation time for the QP solver. Interpolated between these samples, we propose using a standard PI/PID controller to provide intermediate sample control inputs, thus retaining control while the optimization problem is still being solved. The benefit of this arrangement is that we can effectively double or triple the sampling rate of our high-performing optimal controller, using just one or two standard PI/PID control inputs during the optimization period.

Honouring Constraints

The immediate problem is how to deal with constraints which was the primary reason to use MPC in the first place. If we were to simply use the system setpoint as the setpoint for the subservient PI/PID controller then there is no way to enforce these output constraints. While we could de-tune the PI/PID controller to ensure a sluggish response, we would lose the benefit of good control from the MPC controller.

However there is an additional feature we can exploit from the standard MPC calculation: the predicted plant output when calculated with the full set of optimal control moves returned from the MPC. Normally only the first move is used as the control input and the optimisation process repeated. However, if we were to use the entire input vector together with the current state vector and prediction matrices, we could predict what the plant output should do if we were to continue MPC control of the system. Therefore we can use this prediction as the setpoint for our PI/PID controller such that as long as the PI/PID controller follows close to it, we have a much better chance our output constraints will be honoured.

How Good is the Plant Prediction?

Given a perfect model with no model/plant mismatch, we can calculate the prediction for *n* samples spaced every *n* samples to see just how good the prediction is, while also establishing practical limits for the sample spacing *n*. Figure 2 shows the plant output predictions for two cases of *n* for the non-minimum phase SISO system given previously. We can see that as *n* is increased, the prediction falls away from the true plant output, which is expected based on the varying control inputs at each step. However for small values of *n* (i.e. less than around 5) we can assume that the prediction actually follows very closely to the true output thus enabling us to use this prediction as our PI/PID setpoint. What is important to note is that the MPC uses the actual external setpoint, while the interpolated PI/PID controller uses the output predictions of the MPC for its setpoint.

Simulating Computation Time

It is worth noting that no Matlab MPC toolbox we have trialled models the computation time of the controller (including the default jMPC Toolbox configuration), and consequently assumes the calculation of the control input is instantaneous. While this is

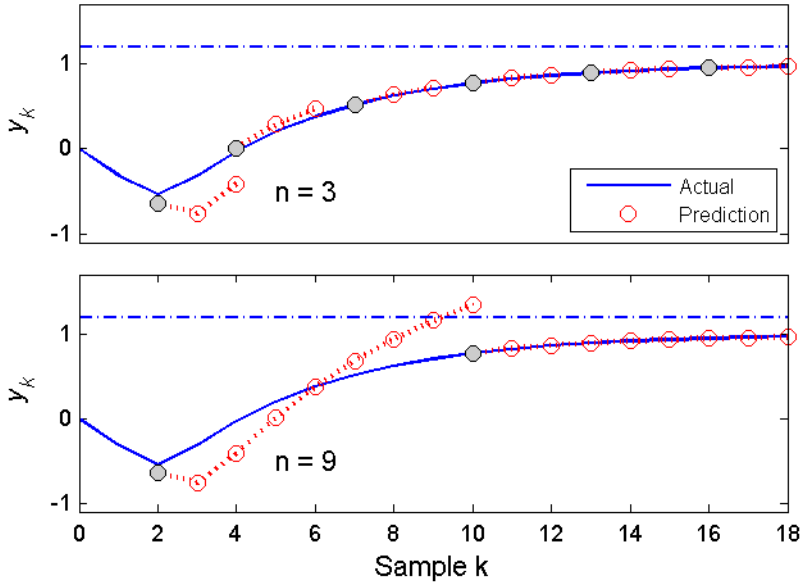

Figure 2: Future plant output predictions (red) for two cases of interpolation horizon *n*: (Upper *n*=3, Lower: *n*=9)

easily overlooked for simulated MPC studies, the result of an industrial MPC controller will actually vary from the "instantaneous MPC" response based on the delay between reading the current plant output, and when a control move is applied.

Interpolated MPC Algorithm

The Interpolated MPC algorithm can best be described using a timing diagram, shown in Figure 3 for an interpolated MPC with *n* of 3.

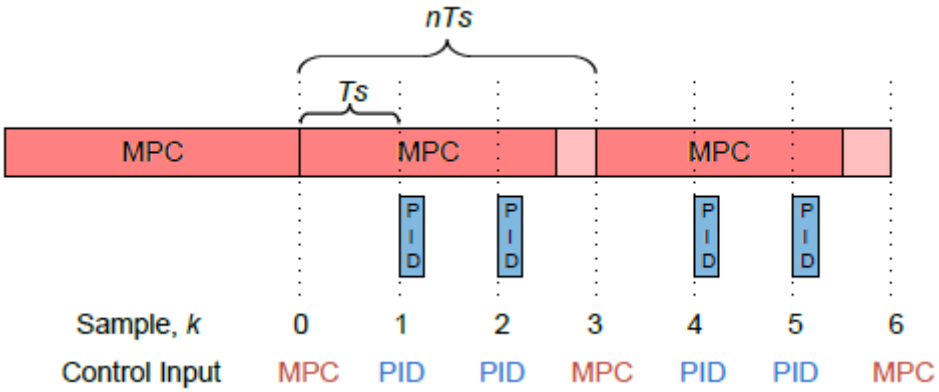

Figure 3 Interpolated MPC timing diagram.

The controller starts with an initial MPC calculation in order to provide the first prediction setpoint and the first control input(s). For simplicity, we assume this takes place before $k = 0$. At the first sample, the control input from the MPC is applied to the plant, and the MPC begins another calculation. For the next two samples ($k = 1,2$) the PI controller uses the first prediction as its setpoint, and maintains the MPC model at each step (discrete state update). At $k = 3$, the MPC has finished its next calculation, and the loop repeats.

The downside to this algorithm is the addition of more controller tuning parameters, specifically the interpolated sample *n*, as well as the three default PI/PID tuning parameters (K, τ_i, τ_d). However from our simulations the interpolated PI/PID parameters modify the plant response in a similar fashion to standard PI/PID control, and at first approximation could be substituted for them. The choice of the *n* should be based on the controller speedup required by the plant, and the best choice is to keep this value as small as possible, and always less than 5.

Theoretical Microcontroller Implementation

Our aim with this research is to develop an MPC algorithm suitable for implementation on standard, low-cost, microcontroller hardware. Such hardware is readily available and suitable for the biotronics/micro-reactor applications referred to earlier in this paper. Using the algorithm described in the previous section, we propose two methods in which this could be implemented in standard architecture.

Interrupt Driven or Parallel Computation

Our first target hardware platform is the Texas Instruments 200MHz Delfino floating point microcontroller. With a single core, but hardware coprocessor for single precision floating point, we can implement and solve the QP using hardware accelerated floating point. In order to provide the PI/PID control, a timer interrupt can call the PI/PID control law (being essentially two dot products), which will only slightly interrupt the MPC calculation. While this implementation is not strictly parallel, the interrupt ensures on time sampling and control action for both controllers. Note that the exceptions to the controller such as bumpless auto-manual transfer are handled outside this algorithm, and the ongoing issues like integral windup are handled by the outer optimal MPC controller.

Our second target hardware platform is the PandaBoard implemented 1GHz dual core ARM Cortex-A9 MPCore. Using the second core for the PI/PID control law as well as general tasks (I/O, communications, etc) enables a fully parallel operation; an MPC dedicated core and a PI/PID dedicated core.

CASE STUDIES

This section will detail two case studies we used for testing the Interpolated MPC algorithm. The first case study will be based on a simpler non-minimum phase SISO system, while an advanced study is applied to the nonlinear CSTR described in a previous section.

Non-minimum Phase SISO with Interpolated MPC

Consider the following second order transfer function with a significant inverse response:

$$\mathrm{G}(s) = \frac{-s + 0.08}{(s + 0.33)\,(s + 0.02)}, \text{ with sample time } T_s = 3$$

The system output y is constrained to ±1.2 and Δu is constrained to ±0.2. A square wave setpoint of ±1 is applied to the system which will challenge the controller to prevent the inverse response from violating the constraints. To create a benchmark for our proposed Interpolated MPC controller (red), we will also simulate this scenario using a standard PI controller (blue) and a standard MPC controller (green) as shown in Figure 4.

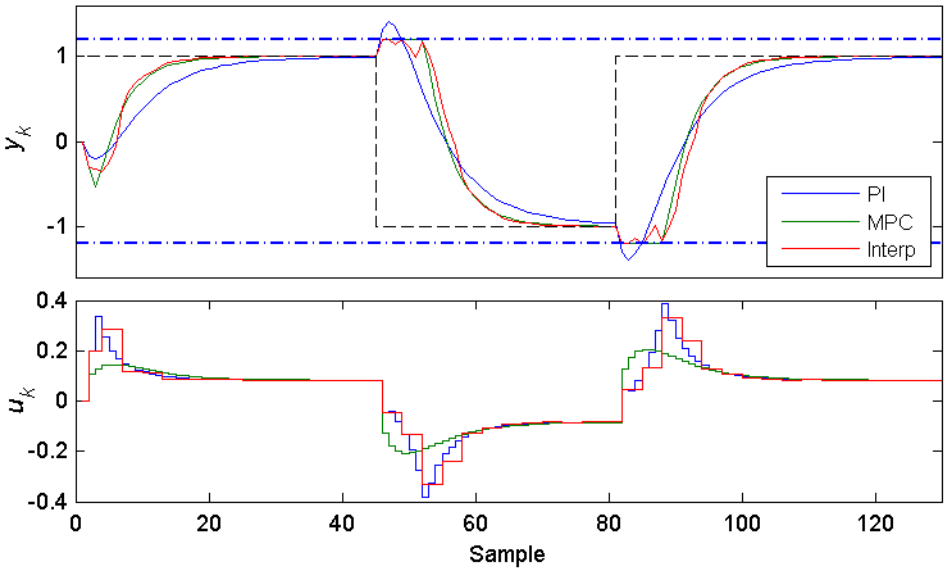

Figure 4: Comparing the proposed Interpolated MPC controller with a standard PI and a standard MPC when applied to a non-minimum phase plant.

The response of the Interpolated MPC controller not only honours the constraints, but behaves very similarly to the original, but computationally expensive, MPC controller. Given that the interpolated MPC uses a sample interval (n) of 3 samples, this corresponds to an effective speedup of three times over the original controller.

Note that the ordinary PI controller does not have any constraint handling ability in contrast to the more advanced controllers. In this case the PI controller is tuned so as to provide a close response to the MPC controller for comparative purposes as it is not the objective for us to detune the PI controller to obey the constraints in this example.

Nonlinear CSTR with Interpolated MPC

The second case study is the nonlinear CSTR presented earlier, which we will control using our Interpolated MPC controller. For this example the interpolated sample time (n) is chosen as 2 samples, and a PID controller with derivative Filter (PIDF) is used in place of the previous PI controller. For this example note that PIDF alone control of this system is considerably difficult and would not normally be adequate.

Figure 5 shows the interpolated controller maintains a very similar control input as the original MPC controller, leading to good control of the reactor temperature. A measured disturbance step change of the feed concentration is applied at sample 130 with just the PIDF controller deviating from the process set point. This indicates the interpolated controller is correctly using the measured disturbance prediction as part of its internal calculation.

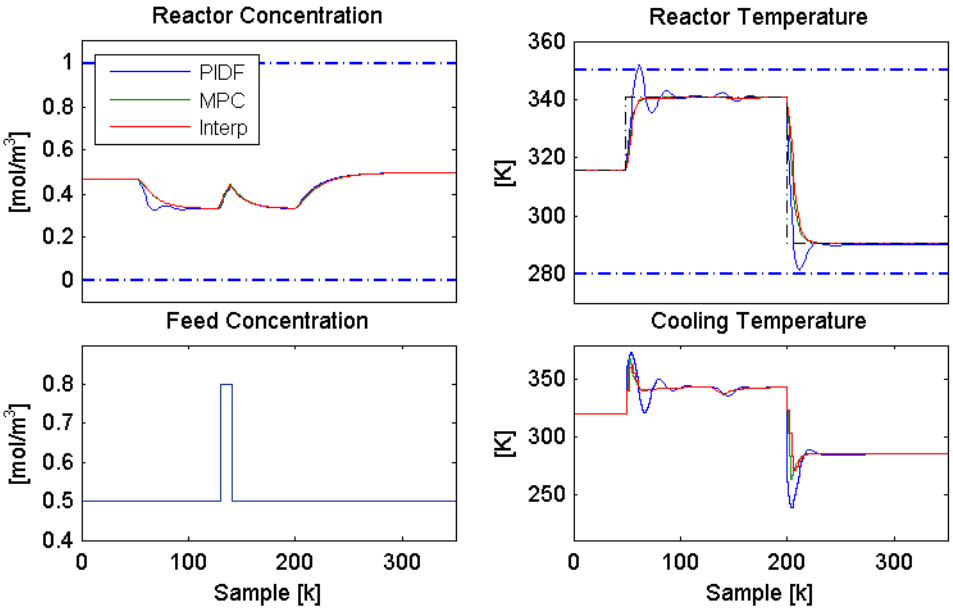

Figure 5 Comparing the proposed interpolated MPC controller with a standard PIDF and a standard MPC for the nonlinear CSTR plant.

Establishing analytical stability guarantees for the interpolated MPC controller is probably intractable for the general nonlinear system, but experience has shown that the control performance (and hence stability) lies somewhere between the pure MPC and PID control as one would probably expect.

CONCLUSIONS

This paper has demonstrated the ability of the Interpolated MPC algorithm to speed up the MPC algorithm by a factor of *n* while honouring constraints and maintaining good control of the process. The two case studies presented are challenging examples for MPC and impossible when constrained for PI/PID control alone, yet our proposed algorithm provides good results for both studies. Furthermore, we have described two candidate embedded systems with implementation details for how the algorithm might be run inside a microcontroller, with an interrupt driven and multi-core design.

ACKNOWLEDGEMENTS

Financial support to this project from the Industrial Information and Control Centre, Faculty of Engineering, AUT University and The University of Auckland, New Zealand is gratefully acknowledged.

REFERENCES

R. Cagienard, P. Grieder, E.C. Kerrigan, and M. Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563-570, 2007.

J. Currie. jMPC Toolbox. A MATLAB MPC Toolbox, 2011. Available from www.i2c2.aut.ac.nz.

J. Currie and D. Wilson. Lightweight Model Predictive Control intended for embedded applications. In *9th International Symposium on Dynamics and Control of Process Systems (DYCOPS)*, Leuven, Belgium, 2010.

P. Dua, K. Kouramas, and E. N. Pistikopoulos. MPC on a chip - Recent advances on the application of multi-parametric model-based control. *Computers and Chemical Engineering*, 32(4-5):754-765, 2008.

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

M. Henson and D. Seborg. *Nonlinear Process Control*. Prentice Hall PTR, 1997.

J. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733-764, July 2003.

J. Rossiter. *Model Based Predictive Control*. CRC Press, 2004.

Petter Tondel, Tor Arne Johansen, and Alberto Bemporad. An algorithm for multi parametric quadratic programming and explicit mpc solutions. *Automatica*, 39(3):489 - 497, 2003.

L.Wang. *Model Predictive Control System Design and Implementation Using MATLAB*. Advances in Industrial Control. Springer, 2009.

David I. Wilson and Brent R. Young. The Seduction of Model Predictive Control. *Electrical & Automation Technology*, pages 27-28, Dec/Jan 2006. ISSN: 1177-2123.

S. Wright. Applying New Optimization Algorithms to Model Predictive Control. *Chemical Process Control-V*, 1997.

Brief Biography of Presenter

Jonathan Currie received his BE (Hons) in Electrical and Electronic Engineering in 2009 from AUT University in New Zealand. Now, he is pursuing a PhD in real-time optimisation of refinery processes, with research focus on optimisation strategies and predictive control.