Preprints of the 20th World Congress
The International Federation of Automatic Control
Toulouse, France, July 9-14, 2017

Reducing Memory Footprints in Explicit Model Predictive Control using Universal Numbers

Deepak Ingole * **Michal Kvasnica** * **Himeshi De Silva** **
John Gustafson **,***

* *Faculty of Chemical and Food Technology, Slovak University of Technology in Bratislava, Slovakia, (e-mail:{deepak.ingole, michal.kvasnica}@stuba.sk).*
** *Department of Computer Science, National University of Singapore, Singapore, (e-mail: himeshi@comp.nus.edu.sg).*
*** *Agency for Science, Technology and Research, Computational Resource Centre, Singapore, (e-mail: john-gustafs@acrc.a-star.edu.sg).*

Abstract: Explicit Model Predictive Control (MPC) is an effective alternative to reduce the on-line computational demand of traditional MPC. The idea of explicit MPC is to pre-compute the optimal MPC feedback law off-line and store it in a form of look-up table which is to be used in on-line phase. One of the main bottlenecks in an implementation of explicit MPC is memory required to store optimal solutions. This limit its applicability to systems with few states, small number of constraints, and short prediction horizons. In this paper, we present a novel way of reducing the memory footprint of explicit MPC solutions. The procedure is based on encoding all data (i.e., the critical regions and the feedback laws) as universal numbers (unums), which can be viewed as a memory-efficient extension of IEEE floating point standard. By doing so, we illustrate that the total memory footprint can be reduced by 80% without losing control accuracy. An additional advantage of proposed approach is, it can be applied on top of existing complexity reduction techniques.

Keywords: Explicit MPC, memory reduction, floating point numbers, universal numbers.

1. INTRODUCTION

Model Predictive Control (MPC) is an advanced control strategy where a Constrained Finite-Time Optimal Control (CFTOC) problem is solved on-line at each sampling interval to obtain the optimal open-loop control sequence. However, only the first input from this sequence is applied to the plant and based on that input, the plant's current state is measured and sent back to the optimizer to compute the next control sequence (Rawlings and Mayne, 2009, Chapter2). A major hurdle in the success of MPC for real-time applications running on embedded platforms is to solve MPC optimization problem within a sample instant. To mitigate the hurdle of on-line computational demand, a multi-parametric programming based approach i.e. *explicit MPC* has been proposed in Bemporad et al. (2002).

In explicit MPC, the optimal control law is pre-computed off-line as a function of all possible initial states. For a large class of MPC problems, such a control law can be shown to take the form of a Piecewise Affine (PWA) function defined over a polyhedral partition in the state space, that maps state measurements onto the optimal control value inputs. Having a pre-computed PWA function at the hand, explicit MPC needs to evaluate the PWA function on-line at each sample instant to compute optimal control actions based on the current state measurement (Bemporad et al., 2002). The major advantages of explicit MPC are: i) by using PWA function, on-line execution-time can be achieved in the range of milli- to microseconds (Oberdieck et al., 2016a), ii) MPC properties like closed-loop stability, feasibility and safety can be verified prior to deploying the control law on hardware, and iii) the PWA function evaluation can be performed on simple embedded hardware like a microcontroller, Programmable Logic Controller (PLC) or Field Programmable Gate Array (FPGA) (Honek et al., 2015).

However, to achieve such a simple and fast implementation, the data related to pre-computed PWA control law needs to be stored on the embedded hardware. Although, this aspect is less addressed in the literature, in fact it plays a crucial role when implementing explicit MPC on embedded devices with low-memory storage capacity and restricted computational power. In the last few years, an effort has been made to reduce the complexity of explicit MPC which is mainly focused on two directions: first, how to make feedback law simple; second, how to reduce the number of bits needed to store data with required accuracy.

The use of a model reduction technique has been proposed in Hovland and Gravdahl (2008) which reduces the

number of regions but with suboptimal control actions. It needs longer control horizons as compared to the horizons needed by original model to obtain good performance. By approximating the optimal MPC feedback, one can obtain less complex but suboptimal feedback function; see e.g. Bemporad and Filippi (2001), Jones and Morari (2009), Johansen and Grancharova (2003). All these techniques lead to a suboptimal control law. The performance-lossless complexity reduction techniques are represented by Optimal Region Merging (ORM) (Geyer et al., 2008), lattice representation (Wen et al., 2009), saturated region clipping (Kvasnica and Fikar, 2012), partial selection (Kvasnica et al., 2012), or region separation (Kvasnica et al., 2013). A good overview of complexity reduction techniques is demonstrated with the Multi-Parametric Toolbox (MPT) in Kvasnica et al. (2015). In all of the cases one can obtain less complex and performance-lossless explicit MPC solution, but the downside is that these techniques are limited to small systems due to the significant pre-computing effort required to solve non-trivial optimization problems.

A common drawback of all the aforementioned approaches is, the data of simplified MPC feedback law needs to be stored as Floating Point (FP) numbers in the IEEE format, 32-bits for single precision and 64-bits for double precision numbers. The bit size of numbers is thus constant regardless of the values they store. One of the way of reducing bit size of the underlying explicit MPC data was presented in Szücs et al. (2011) where, the authors have used Huffman encoding (Knuth, 1985) to compress some of the data (specifically, integer indices to a set of unique half-spaces). The procedure can thus be viewed as a variable-size encoding. Since there is a one-to-one correspondence between original data and their compressed counterparts, the compressed feedback law exhibits the same properties (e.g. control performance, closed-loop stability and constraint satisfaction) as the original one. An alternative was presented by Suardi et al. (2016) where the authors have proposed use of low precision arithmetic.

This paper aims at reducing the memory footprint of explicit MPC by using *universal numbers* (unums) (Gustafson, 2015) to represent the data associated with the optimal MPC feedback law. This allows one to use an explicit MPC for the number of systems that would otherwise be excluded due to the high complexity of resulting explicit controllers; that is, due to the lack of available memory or powerful computing devices to make point location algorithms faster. A key idea of unums is to store a real number with a variable bit length format using six subfields: sign bit, exponent, fraction, uncertainty bit, exponent size, and fraction size. Basically, unum is a superset of IEEE 754 floating point format (Muller et al., 2009) that tracks whether a number is an exact float or lies in the open interval between two exact floats. Compared to the standard floating point formats; the variable size in unum offers an ability to change its representative range, precision, and the uncertainty bit indicates the exactness of represented value. Thus, unums use fewer bits, obey algebraic laws, and do not require rounding, overflow, and underflow for proper operations (Gustafson, 2015, Chapter 3).

2. PRELIMINARIES AND PROBLEM STATEMENT

2.1 Explicit MPC

Consider the class of discrete time Linear Time-Invariant (LTI) systems

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^l$ is the control input, $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times l}$ and the pair (A, B) is stabilizable. State and input variables are subject to the polytopic constraints $x \in \mathcal{X} \subseteq \mathbb{R}^n$, $u \in \mathcal{U} \subseteq \mathbb{R}^l$ where $\mathcal{X}$ and $\mathcal{U}$ are polyhedral sets containing the origin in their respective interior. The constrained finite time optimal control problem for the LTI system in (1) is

$$\min_{U_N} x_N^T P x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \tag{2a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1, \tag{2b}$$

$$x_k \in \mathcal{X}, \qquad k = 0, \dots, N-1, \tag{2c}$$

$$u_k \in \mathcal{U}, \qquad k = 0, \dots, N-1, \tag{2d}$$

$$x_N \in \mathcal{X}_f, \tag{2e}$$

$$x_0 = x(t), \tag{2f}$$

where $Q \in \mathbb{R}^{n\times n}$, $R \in \mathbb{R}^{l\times l}$ and $P \in \mathbb{R}^{n\times n}$ are the weighting matrices, with conditions $Q \succeq 0$ and $P \succeq 0$ to be positive semidefinite, and $R \succ 0$ to be positive definite. We denote by N the prediction horizon, x_{k+1} as the vector of predicted states at time instant k, $U_N = \{u_0, \dots, u_{N-1}\}$ as the sequence of control actions, x_0 as the initial conditions, and $\mathcal{X}_f$ as the polyhedral constraint set for the terminal state x_N.

It is well known, see, e.g. (Borrelli et al., 2011, Chapter 12), that by using the substitution $x_k = A^k x_0 + \sum_{j=0}^{k-1} \left(A^{k-1-j}B\right) u_j$ the MPC problem (2) can be translated into a multi-parametric quadratic problem (mp-QP) of the form

$$\min_{U_N} U_N^T H U_N + x_0^T F U_N + x_0^T Y x_0 \tag{3a}$$

$$\text{s.t. } GU_N \leq w + Wx_0, \tag{3b}$$

where matrices $H \in \mathbb{R}^{lN\times lN}, F \in \mathbb{R}^{n\times lN}, Y \in \mathbb{R}^{n\times n}, G \in \mathbb{R}^{q\times lN}, w \in \mathbb{R}^q, W \in \mathbb{R}^{q\times n}$ and q is a number of inequalities.

Furthermore, as demonstrated e.g. by Bemporad et al. (2002), the mp-QP (3) admits a closed-form solution as a function $\kappa : \mathbb{R}^n \to \mathbb{R}^{lN}$ that maps the initial conditions x_0 onto the sequence of optimal control inputs $U_N^\star$, i.e., $U_N^\star = \kappa(x_0)$. Moreover, provided that $H \succ 0$ (which is satisfied once $R \succ 0$, $Q \succeq 0$, $P \succeq 0$), κ is a continuous piecewise affine (PWA) function over polyhedral critical regions:

$$\kappa(x) = \begin{cases} L_1 x + g_1 & \text{if } x \in \mathcal{R}_1 \\ \vdots & \\ L_M x + g_M & \text{if } x \in \mathcal{R}_M \end{cases} \tag{4}$$

where

$$\mathcal{R}_i = \{x \in \mathbb{R}^n \mid Z_i x \leq z_i\} \quad i = 1, \dots, M \tag{5}$$

are the polyhedral regions with $Z_i \in \mathbb{R}^{c_i\times n}$, $z_i \in \mathbb{R}^{c_i}$ describing the half-spaces of the i-th region with c_i being the number of half-spaces of the i-th region, and $L_i \in \mathbb{R}^{lN\times n}$,

$g_i \in \mathbb{R}^{lN}$ are locally optimal gains. Moreover, M denotes the total number of regions. The data Z_i, z_i, L_i, g_i can be computed e.g. by the Multi-Parametric Toolbox (Herceg et al., 2013), the Hybrid toolbox (Bemporad, 2004), or the POP toolbox (Oberdieck et al., 2016b).

Once the PWA function κ in (4) is constructed off-line, the on-line implementation of MPC boils down to evaluation of κ for a known value of $x_0 = x(t)$ (obtained either by direct measurement or by estimation). Such an evaluation is fast and simple since it does not involve divisions. Only multiplication and addition are required to identify $U_N^\star$ for a given value of x_0.

2.2 Problem Statement

To implement (4) as a feedback controller, one first needs to store all its real-valued data (i.e., vectors/matrices Z_i, z_i, L_i, g_i) in the memory of control hardware. The total memory footprints of explicit solution in (4) expressed as floating point numbers can be compactly given by

$$\mathcal{S}(\kappa) = MlN(n+1) + \sum_{i=1}^{M} c_i(n+1), \tag{6}$$

where first part represents the size of all L_i, g_i pairs [1] (which have constant dimensions for all regions), and the second part represents memory footprints of all polyhedral regions (with c_i being the number of half-spaces [2] of the i-th region). The bit size of κ is then $\mathcal{B}(\kappa) = b\mathcal{S}(\kappa)$ where either $b = 32$ or $b = 64$, depending either on single precision or on double precision floating point numbers are used.

Our objective is to reduce the bit size of a given explicit controller in (4) by devising a more memory efficient representation of floating point numbers contained in the real-valued vectors/matrices Z_i, z_i, L_i, g_i. This can be achieved by representing each floating point number as a universal number with a variable size of its bit code. In other words, instead of using a constant value of b as the bit size of each floating point number, each number is represented by b_j bits with $b_j \leq b$. This needs to be done in such a way that the variable-sized bit codes encode the same information as fixed size floating point numbers, i.e., the encoding/decoding is done in a *performance-lossless* fashion. In the next section we show how this can be achieved using universal numbers.

3. UNIVERSAL NUMBERS

In this section, we will use different colors to specify various bits in a number presentation.

3.1 IEEE 754 Floating Point Format

The IEEE 754 binary floating point is the most commonly used representation for real numbers and it comprise of three fields: sign (s), exponent (e) (true exponent + bias), and mantissa (m). A binary FP number can be encoded using four formats: half precision (16-bits), single precision (32-bits), double precision (64-bits), and quad precision (128-bits). For detailed information about the number conversions (real $\rightleftarrows$ FP), arithmetic operations, and exception handling see, e.g. Zuras et al. (2008); Muller et al. (2009). The IEEE 754 FP format has many advantages but it comes with trade-offs like rounding errors, overflow, underflow, and the "one size fits all" data format using the same number of bits $(16, 32, 64$ or $128)$ even if it is not necessary (Goldberg, 1991).

3.2 Universal Number Format

Unum format is a superset of IEEE 754 FP formats, with six sub-fields instead of three. Fig. 1 shows the general representation of unum format. The left three fields are like IEEE FP format but with unums these fields have better rules for handling special numbers like Not-a-Number (NaN) and infinity. Also, unums can handle overflow, underflow, and rounding in a much better way than FP. The description of each field is given below:

1. Sign (s): in unum it is same as the sign bit in floating point numbers. For positive numbers, sign is 0 and for negative numbers it is 1.
2. Exponent (e): in unum it is like exponent in the floating point number but its bit length is specified by exponent size denoted by es; see below.
3. Mantissa (m): like the exponent, mantissa in unum is same as in floating point numbers but its length depends on the number of bits specified by mantissa size denoted by ms; see below.
4. Uncertainty bit (ub): this bit in the unum is used to indicate whether the number is exact or in an interval. It is 0 if unum is exact and 1 if unum is in the open interval between two exact unums.
5. Exponent size (es): the fields es and ms are self-descriptive and their lengths offset by 1. The number of bits in this field depends on "how many bits we want to allocate to specifyes" i.e. the size of exponent size (ess). For example, exponent value $= 1111_2$ in binary needs 4-bits i.e. exponent size $(es) = 4$, which is 100_2 in binary, and $ess = 3$-bits (number of bits needed to express (es) of 4-bits). The number of bits in this field is in the range of 1-bit to 2^{ess}-bits.
6. Mantissa size (ms): the number of bits in this field depends on "how many bits we want to allocate to specify ms" i.e. the size of mantissa size (mss). The number of bits in this field is in the range of 1-bit to 2^{mss}-bits.

The exponent sizes in IEEE FP format are 5 (101_2) for half precision, 8 (1000_2) for single precision, 11 (1011_2) for double precision, and 15 (1111_2) for quad precision. Hence, four bits suffice to cover all IEEE FP formats. There is always at least one exponent bit and one mantissa bit. Therefore, we keep an offset of one in the last two fields of unum i.e. $es - 1$ and $ms - 1$. Because of this offset in original exponent sizes, IEEE exponent sizes can be represented as (100_2), (111_2), (1010_2), and (1110_2). In the unum, four bits would be enough to specify any exponent size ranging from 2^0-bit (fixed point format) to 2^4-bits (more than IEEE FP quad format). As with the exponent

[1] In fact, if the MPC controller is implemented in a receding horizon fashion, then only the first component of $U_N^\star$, i.e., $u_0^\star$, is required. In such a case the matrices L_i, g_i can be truncated to just the first l rows and the first part in (6) becomes $Ml(n+1)$.

[2] At this point we assume that only the non-redundant half-spaces are stored, i.e., $c_i \leq q$ where q is the number of constraints in (3b).

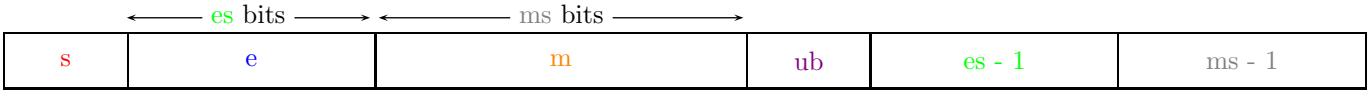

Fig. 1. General representation of the universal number format with six sub-fields.

sizes, mantissa sizes in IEEE FP format are 10 (1010_2), 23 (10111_2), 52 (101100_2), and 112 (1110000_2). So, seven bits are enough to cover all IEEE mantissa sizes (again, offset by one). This covers mantissa sizes from 2^0-bit to 2^7-bits. The number of bits needed to specify the exponent size size (zero to four) and mantissa size size (zero to seven) in unum are called *ess* and *mss* respectively, and that pair is called the *Environment* or env$\{ess, mss\}$ and unum with specific *Environment* is denoted as unum$\{ess, mss\}$ e.g. unum$\{2, 2\}$. User is free to define this pair based on the needs of an application. The *Environment* can be as small as $\{0, 0\}$ or as large as computer memory and speed permits (Gustafson, 2015, Chapter 4).

In the unum format, the set of last three fields (*ub*, *es*, and *ms*) is called the *utag* and the number of bits in *utag* is called *utagsize* which is equal to $1+ess+mss$. As described above, using env$\{4, 7\}$ we can achieve the superset of IEEE FP format with the *utagsize* equal to $1+4+7=12$. Also, the minimum number of bits in a unum is $3+utagsize$ and the maximum is $2+ess+mss+2^{ess}+2^{mss}$. At this stage, one might wonder: *How can the unum approach help to reduce memory as compared to floating point, if it adds more bits to the format?* the main reason is that unum allows us to specify exponent and mantissa using fewer bits compared to the exponent and mantissa in IEEE FP formats.

3.3 Floating Point to Universal Number Conversion

Example 1: Represent a constant π in unum format. The constant π is approximated to 11-decimal accuracy as 3.1415926535 (Finch, 2003, Chapter 1). In general, a floating point approximation of a real value can be expressed as (Gustafson, 2015, Chapter 3)

$$(-1)^s \times \begin{cases} 2^{2-2^{es-1}} \times \left(\frac{m}{2^{ms}}\right) & \text{if } e = \text{all 0 bits,} \\ \infty & \text{if } e \text{ and } m \text{ are all 1 bits,} \\ 2^{1+e-2^{es-1}} \times \left(1+\frac{m}{2^{ms}}\right) & \text{otherwise.} \end{cases} \quad (7)$$

In binary IEEE 754 double precision, the constant π looks as follows:

0	10000000000	1001001000011111101101010100010001000010110100011

In this representation we used 64-bits to store the approximated value of π. Here, we can see that even after using 64-bits we are not getting the exact value of π and moreover we wasted many bits.

Having the IEEE 64-bit representation of π at the hand, one can be proceed to convert it to unum format. Assume that we want to have a 4-bit exponent and a 4-bit mantissa for the value of π, in unum it can be obtained by setting env$\{2, 2\}$.

Similar to the FP expression, unum equivalent real value of unum can be expressed as

$$(-1)^s \times \begin{cases} 2^{2-2^{es-1}} \times \left(\frac{m}{2^{ms}}\right) & \text{if } e = \text{all 0 bits,} \\ \infty & \text{if } e,\ m,\ es, \text{ and } ms \text{ have all bits} = 1, \\ 2^{1+e-2^{es-1}} \times \left(1+\frac{m}{2^{ms}}\right) & \text{otherwise.} \end{cases} \quad (8)$$

This expression is the improved version of (7) which does not waste the huge number of bits on *NaN*. Using unum format the value of constant π can be represented in only 11-bits as given below

0	1	1001	1	00	11

It is interesting to note that only 11-bits are used to store the value of π which saved 53-bits compared to the IEEE double precision format. Also, we can see that *ub* bit is 1 which means that the number is inexact and the real value is an interval. To obtain interval values we use the Unit of Least Precision (*ULP*) which is equal to the difference between exact values in bit format that differs by one unit in the last place. Mathematical meaning of the unum bit format can be obtained from (8) which gives the value of π as an open interval $(3.125, 3.25)$.

3.4 Universal Number Arithmetic

Universal number arithmetic is somewhat similar to interval arithmetic but, unum has additional complexity due to the open versus closed endpoints, dynamic exponent, mantissa sizes, and correct handling of math operations (add, sub, mul, and div) that exceed the limits of unums in a particular *environment*.

Addition/Subtraction: The addition of two real numbers Z_1 and Z_2 can be perform by obtaining their *ubounds* like $[a, b]$ and $[c, d]$ and then the addition of these two *ubounds* is straightforward i.e., $[a+c, b+d]$. But, enough care is needed to add open and closed intervals. If we want to add the open interval $(-\infty, 0)$ to ∞, the correct answer in unum is ∞ because the left endpoint "$(-\infty$" indicates some finite value. If we add ∞ to any finite value it results in ∞. For addition of *ubounds* there are several rules and special cases which are described in (Gustafson, 2015, Chapter 9). Similar to addition, subtraction of *ubounds* is carried out by simply adding the first argument to the negative of second argument, i.e. $Z_1 - Z_2 = Z_1 + (-Z_2)$.

Multiplication: For real intervals, if both *ubounds* are closed intervals multiplication is simple as follows:

$$[a, b] \times [c, d] = [\min \mathcal{N}, \max \mathcal{N}] \quad (9)$$

where $\mathcal{N} = (a \times c, a \times d, b \times c, b \times d)$. But, for unbounded intervals (9) does not hold. To tackle with the multiplication of unbounded intervals we consider the multiplication table for left and right endpoints defined in (Gustafson, 2015, Chapter 10). In a multiplication of conventional

interval arithmetic the case $0 \times \pm\infty$ is undefined which results in NaN but, in unum it has more information about the endpoints and we get the following cases:

- The result of $\infty \times (a, b)$ is $[-\infty, \infty]$ and not NaN, where a and b are the endpoints of opposite sign.
- The result of $0 \times \text{inexact}\,\infty$ is 0 and $\text{inexact}\,0 \times \infty$ is ∞.
- The result of $\text{inexact}\,0 \times a$ is a nonzero value, where a is any finite nonzero number.
- The result of $\text{inexact}\,0 \times \text{inexact}\,\infty$ is the entire positive real number line.

For the multiplication of negative-negative and positive-negative numbers, see (Gustafson, 2015, Chapter 10).

Compare Operators: The comparison $Z_1 \leq Z_2$ or $(a, b) \leq (c, d)$ is equivalent to checking $b < c$.

3.5 Universal Number MATLAB® *Library*

To encode the data of explicit optimizer in (4) (i.e., the floating point numbers contained in vectors/matrices Z_i, z_i, L_i, g_i) in the presented unum format, we have developed [3] an open-source MATLAB® library called `munum`. It supports every *environment* from $\{0, 0\}$ to $\{4, 7\}$ which covers all four IEEE 754 floating point formats. Moreover, the library implements basic primitives (such as addition, multiplication and comparison) of unums that can be used to implement the unum-encoded explicit optimizer (4). On the top of these basic math operations, library also supports addition, subtraction, and multiplication of matrices and vectors. These can be used directly in the point location algorithms used in explicit MPC.

4. EXAMPLE: DOUBLE INTEGRATOR

We illustrate the efficiency of unum-based memory reduction technique with the following example. Consider the discrete-time version of the double integrator system

$$x_{k+1} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} u_k, \tag{10}$$

with the sampling time $T_s = 0.05$s. Here, x_1, x_2 and u is position, speed and input acceleration, respectively. This system is subject to state and input constraints as follows

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix} \text{ and } -1 \leq u \leq 1. \tag{11}$$

The MPC problem (2) was solved parametrically for the set of different prediction horizons (listed as the first column in table 1) with $Q = I_{2\times 2}$, $R = 1$, the terminal penalty $P = 5I_{2\times 2}$, and $\mathcal{X}_f = \{x_N \mid [-1 \quad -1]^\top \leq x_N \leq [1\ 1]^\top\}$. The explicit MPC controller is constructed using the Multi-Parametric Toolbox (MPT). Finally, the data of explicit optimizer in (4) were encoded into the unum format using the MATLAB® library presented in Section 3.5.

For each value of the prediction horizon, the memory footprints taken by floating point and unum number format were recorded. FP memory footprints were obtained by (6) considering double precision format and unum with two environments i.e., $\{2, 2\}$ and $\{3, 2\}$. With env$\{2, 2\}$ every number can be stored in less than or equal to 14-bits, which means that to store the explicit control law we will need at most 21.87 % of the memory compared to the 64-bit floating point format. The env$\{2, 2\}$, is sufficient to get the required accuracy for the double integrator problem but even if we use a higher *environment* like env$\{3, 2\}$ then the memory required will still be no more than 29.68 %.

Particular results for each value of the prediction horizon are shown in table 1. As can be observe, the required memory space can be reduced by $\approx$ 80% when using the $\{2, 2\}$ environment or by $\approx$ 78% when choosing the $\{3, 2\}$ environment. Interesting thing to notice is that the byte size of the unum-encoded solution for $N = 40$ (75.8 kilobytes and 3359 regions) is roughly equal to the size of floating point representation for $N = 17$ (75.3 kilobytes and 643 regions). It follows that by using unums one can fit more data into the same space and thus can be able to use larger prediction horizons in explicit MPC.

5. CONCLUSIONS

In this paper, we have shown, how to reduce the memory footprints of explicit MPC feedback law for the implementation on control devices with limited storage space. The approach is based on representing the controller data by unum format which takes fewer bits as compared to the "one-size fits all" IEEE 754 floating point data format. A MATLAB® library was developed to demonstrate the feasibility of unums to reduce the memory size without losing control accuracy. Further, the unum-based explicit MPC is developed and employed for the double integrator control problem with varying prediction horizons. The resulting memory footprints are compared with those of the floating point-based explicit MPC approach. The result shows proposed memory reduction technique can reduce memory footprints by 80 % as compared to the floating point approach. With the use of appropriate *environment* in unums, it is possible to use longer prediction horizons while achieving small memory footprints.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO). Deepak Ingole and Michal Kvasnica gratefully acknowledge the contribution of the Slovak Research and Development Agency under the project APVV 0551-11 and the contribution of the Scientific Grant Agency of the Slovak Republic under the grant 1/0403/15. We would like to thank Dr. Simon Byrne from the Department of Statistical Science, University College, London, United Kingdom for helpful discussions.

REFERENCES

Bemporad, A. (2004). Hybrid Toolbox - User's Guide.

Bemporad, A. and Filippi, C. (2001). Suboptimal explicit MPC via approximate multiparametric quadratic programming. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 5, 4851–4856. IEEE.

[3] Available at `https://bitbucket.org/kvasnica/munum`

Table 1. Comparison of memory footprints for floating point (64-bit) and universal number based explicit MPC for double integrator system.

N	Number of	Total	Number of Bits			Memory[kB]			Memory Savings [%]	
	Regions	Numbers	FP	unum{2, 2}	unum{3, 2}	FP	unum{2, 2}	unum{3, 2}	unum{2, 2}	unum{3, 2}
2	13	195	12480	2459	2674	1.52	0.30	0.32	80.29	78.57
5	67	999	63936	12717	13864	7.80	1.55	1.69	80.10	78.31
7	123	1833	117312	23311	25396	14.32	2.84	3.10	80.12	78.35
10	237	3543	226752	44767	48759	27.67	5.46	5.95	80.25	78.49
12	333	4983	318912	62756	68352	38.92	7.66	8.34	80.32	78.56
15	507	7593	485952	95491	104002	59.32	11.65	12.69	80.34	78.59
17	643	9633	616512	120901	131670	75.25	14.75	16.07	80.38	78.64
20	877	13143	841152	163991	178651	102.67	20.01	21.80	80.50	78.76
23	1147	17193	1100352	213674	222800	134.32	26.08	28.41	80.58	78.84
25	1347	20193	1292352	250402	272815	157.75	30.56	33.30	80.62	78.89
30	1917	28743	1839552	354772	386613	224.55	43.30	47.19	80.71	78.98
40	3359	50373	3223872	621237	677107	393.53	75.83	82.65	80.73	78.99

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E.N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3–20.

Borrelli, F., Bemporad, A., and Morari, M. (2011). Predictive control for linear and hybrid systems. *Cambridge February*.

Finch, S. (2003). *Mathematical constants*. Cambridge University Press.

Geyer, T., Torrisi, F.D., and Morari, M. (2008). Optimal complexity reduction of polyhedral piecewise affine systems. *Automatica*, 44(7), 1728–1740.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1), 5–48.

Gustafson, J. (2015). *The End of Error: Unum Computing*. CRC Press.

Herceg, M., Kvasnica, M., Jones, C., and Morari, M. (2013). Multi-parametric toolbox 3.0. In *Proceedings of the European control conference*, EPFL-CONF-186265.

Honek, M., Kvasnica, M., Szũcs, A., Šimončič, P., Fikar, M., and Rohaľ-Ilkiv, B. (2015). A low-complexity explicit mpc controller for afr control. *Control Engineering Practice*, (42), 118–127.

Hovland, S. and Gravdahl, J.T. (2008). Complexity reduction in explicit MPC through model reduction. *IFAC Proceedings Volumes*, 41(2), 7711–7716.

Johansen, T.A. and Grancharova, A. (2003). Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Transactions on Automatic Control*, 48(5), 810–815.

Jones, C. and Morari, M. (2009). Approximate explicit MPC using bilevel optimization. In *Control Conference (ECC), 2009 European*, 2396–2401. IEEE.

Knuth, D. (1985). Dynamic Huffman Coding. *J. Algorithms*, 6(2), 163–180.

Kvasnica, M. and Fikar, M. (2012). Clipping-based complexity reduction in explicit MPC. *IEEE Transactions on Automatic Control*, 57(7), 1878–1883.

Kvasnica, M., Hledík, J., and Fikar, M. (2012). Reducing the memory footprint of explicit MPC solutions by partial selection. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 4537–4542. IEEE.

Kvasnica, M., Hledík, J., Rauová, I., and Fikar, M. (2013). Complexity reduction of explicit model predictive control via separation. *Automatica*, 49(6), 1776–1781.

Kvasnica, M., Holaza, J., Takács, B., and Ingole, D. (2015). Design and verification of low-complexity explicit MPC controllers in MPT3. In *Control Conference (ECC), 2015 European*, 2595–2600. IEEE.

Muller, J., Brisebarre, N., De Dinechin, F., Jeannerod, C., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., and Torres, S. (2009). *Handbook of floating-point arithmetic*. Springer Science & Business Media.

Oberdieck, R., Diangelakis, N., Nascu, I., Papathanasiou, M., Sun, M., Avraamidou, S., and Pistikopoulos, E. (2016a). On multi-parametric programming and its applications in process systems engineering. *Chemical Engineering Research and Design*.

Oberdieck, R., Diangelakis, N.A., Papathanasiou, M., Nascu, I., and Pistikopoulos, E. (2016b). Pop–parametric optimization toolbox. *Industrial & Engineering Chemistry Research*, 55(33), 8979–8991.

Rawlings, J.B. and Mayne, D.Q. (2009). *Model predictive control: Theory and design*. Nob Hill Pub.

Suardi, A., Longo, S., Kerrigan, E.C., and Constantinides, G. (2016). Explicit MPC: Hard constraint satisfaction under low precision arithmetic. *Control Engineering Practice*, 47, 60–69.

Szücs, A., Kvasnica, M., and Fikar, M. (2011). A memory-efficient representation of explicit MPC solutions. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 1916–1921. IEEE.

Wen, C., Ma, X., and Ydstie, B.E. (2009). Analytical expression of explicit MPC solution via lattice piecewise-affine function. *Automatica*, 45(4), 910–917.

Zuras, D., Cowlishaw, M., Aiken, A., Applegate, M., Bailey, D., Bass, S., Bhandarkar, D., Bhat, M., Bindel, D., and Boldo, S. (2008). IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, 1–70.