Approximate LASSO Model Predictive Control for Resource Constrained Systems

Yun Wu, João F. C. Mota, Andrew M. Wallace
The School of Engineering and Physical Sciences
Heriot-Watt University
Edinburgh, UK
{y.wu, j.mota, a.m.wallace}@hw.ac.uk

***Abstract*—LASSO MPC is a popular method for solving optimal control problems within a receding horizon. It is, however, challenging to deploy LASSO MPC on resource constrained systems, such as embedded platforms, due to the intensive memory usage and computational cost as the horizon length is extended. By exploiting a reduced precision, approximation technique applied to Proximal Gradient Descent (PGD), we demonstrate an implementation on a resource constrained, reconfigurable device, such as a Field Programmable Gate Array (FPGA). Our experiments show equivalent performance to a high-precision optimisation solver, but with significant improvements to both logic cost and memory bandwidth, up to 60% and 80% reduction respectively, with up to 70% power savings.**

***Index Terms*—MPC, LASSO, Proximal Gradient Descent, Approximate Computing, FPGA**

I. Introduction

Model Predictive Control (MPC) is a popular technique to solve optimal control problems in discrete time. Its robustness, stability, and theoretical guarantees have made it widely adopted by industry [1]. Given a dynamic model of a system and an estimate of its current state, MPC computes the next states and necessary control inputs by minimizing a cost function that balances the achievement of a desired state in a predefined time-horizon with the energy required. As energy is often expressed in a quadratic form, MPC typically entails the application of (nonzero) input signals to the system at all time steps.

To reduce the number of times that inputs are applied, the work in [2] proposed *LASSO MPC* which, inspired by results on sparsity, regularizes the energy term with an ℓ_1-norm penalty. This leads to sparse input signals. Despite many desirable features, (LASSO) MPC requires an iterative procedure to solve an optimization problem at each time step. This not only makes execution in real-time challenging (as the number of required iterations is unknown a priori), but also demands significant memory and computation, which can be limiting on resource-constrained systems, such as embedded hardware.

In such systems, one often has to trade-off accuracy for power savings by using approximate computing (AC) techniques [3]. One example is reduced precision (RP), in which data is represented with fewer bits than desired throughout the entire computational stack. As arbitrary precision arithmetic is not supported in many modern processors, a flexible hardware architecture enabling RP, such as reconfigurable platform using an FPGA, is demanded.

Work supported by EPSRC Grant number EP/S000631/1 and the MOD University Defence Research Collaboration (UDRC) in Signal Processing.

Our goal. We aim to implement an efficient solution to LASSO MPC and deploy it on an FPGA with the goal of achieving real-time performance [4]. Our optimization algorithm of choice is proximal gradient descent (PGD) [5]. This is sufficiently tractable to allow us to analyze the effects of different RP strategies on accuracy, logic and memory resources, and power consumption.

Contributions. We summarize our contributions as follows:

- We apply PGD to LASSO MPC which, to the best of our knowledge, has never been done, and analyze its performance. The application of accelerated versions of PGD [5] should be immediate.
- We introduce an Approximate Core (AC) synthesis infrastructure.
- Using our infrastructure, we then conduct a detailed study of the effects of the AC strategy on our algorithm.

II. Background

We briefly explain MPC, LASSO MPC, and then review different number representations and their use in RP computing.

State-space models. We consider state-space representations of linear time-invariant (LTI) discrete systems. At each time, t, such systems are completely described by a state vector $x[t] \in \mathbb{R}^n$, which is known to evolve as

$$x[t+1] = Ax[t] + Bu[t]\,, \qquad t = 0, 1, \dots\,, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are given matrices (assumed known), and $u[t] \in \mathbb{R}^m$ is the set of inputs at time t.

Model predictive control (MPC). Given a finite time-horizon $T \in \mathbb{N}$, a desired final state $x_f \in \mathbb{R}^n$, and an estimate of the current state $x_0 \in \mathbb{R}^n$, MPC attempts to compute a minimal energy state-trajectory such that the final state $x[T]$ is as close as possible to the desired one, x_f. This can be formulated as an optimization problem:

$$\begin{aligned} \min_{\overline{x},\overline{u}} \quad & F(x[T]) + \textstyle\sum_{t=0}^{T-1} \ell(x[t], u[t]) \\ \text{s.t.} \quad & x[t+1] = Ax[t] + Bu[t]\,, \quad t = 0, \dots, T-1 \\ & x[0] = x_0\,, \end{aligned} \tag{2}$$

978-1-7281-3810-7/20/$31.00 ©2020 IEEE

where $(\overline{x}, \overline{u}) := (\{x[t]\}_{t=0}^{T}, \{u[t]\}_{t=0}^{T-1}) \in \mathbb{R}^{n(T+1)} \times \mathbb{R}^{mT}$ is the optimization variable, $F : \mathbb{R}^n \to \mathbb{R}$ a function that penalizes deviations of the final state $x[T]$ from the desired one, and $\ell : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ a function that measures "energy" at each time instant. Notice that the first set of constraints is exactly (1), and the second set reflects the current state.

The functions F and ℓ are often quadratic forms. An example assuming $x_f = 0$ would be

$$F(x) = x^\top P x \tag{3a}$$
$$\ell(x, u) = x^\top Q x + u^\top R u\,, \tag{3b}$$

where $P, R \succ 0$ are positive definite matrices and $Q \succeq 0$ is positive semidefinite. With this choice, the variable $\overline{x}$ in (2) can be eliminated. To see why, first write (1) as [6, Ch. 8]

$$\underbrace{\begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[T] \end{bmatrix}}_{=:\overline{x}} = \underbrace{\begin{bmatrix} I_n \\ A \\ \vdots \\ A^T \end{bmatrix}}_{=:\overline{A}} x_0 + \underbrace{\begin{bmatrix} 0 & \cdots & 0 \\ B & \cdots & 0 \\ AB & \ddots & 0 \\ A^{T-1}B & \cdots & B \end{bmatrix}}_{=:\overline{B}} \underbrace{\begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[T-1] \end{bmatrix}}_{=:\overline{u}}, \tag{4}$$

where I_n is the identity matrix in $\mathbb{R}^n$. Placing the resulting equation into the cost of (2) and manipulating, we obtain

$$\underset{\overline{u}}{\text{minimize}} \;\; \frac{1}{2}\overline{u}^\top(\overline{B}^\top \overline{Q}\,\overline{B} + \overline{R})\overline{u} + (\overline{B}^\top \overline{Q}\,\overline{A}\,x_0)^\top \overline{u}\,, \tag{5}$$

where $\overline{R} := I_T \otimes R$, and $\overline{Q}$ is the diagonal concatenation of $I_T \otimes Q$ and P ($\otimes$ denotes the Kronecker product). Problem (5) is unconstrained quadratic, and thus has a closed-form solution. In MPC [6], whenever (5) is solved, only the first input $u[0]$ is applied to the system, the resulting state is measured, and (5) is solved again using x_0 as the current state.

LASSO MPC. Although (5) has a closed-form solution, it typically yields dense (i.e., non-sparse) inputs, which can lead to over-actuated systems. To encourage sparse inputs, [2] proposed LASSO MPC which adds an ℓ_1-norm penalty $\lambda\|\overline{u}\|_1$, with $\lambda > 0$, to (5) [or, equivalently, $\lambda\|u\|_1$ to (3b)]. Noticing that the objective of (5) can be written as $\frac{1}{2}\|H\overline{u} - y\|_2^2 - \frac{1}{2}\|y\|_2^2$, with $H := (\overline{B}^\top \overline{Q}\,\overline{B} + \overline{R})^{1/2}$ and $y := -H^{-1}\overline{B}^\top \overline{Q}\,\overline{A}x_0$, the resulting problem is

$$\underset{\overline{u}}{\text{minimize}} \;\; \frac{1}{2}\|H\overline{u} - y\|_2^2 + \lambda\|\overline{u}\|_1\,, \tag{6}$$

which has format of LASSO [7]. However, the matrix H in (6) is square. So, instead of regularizing the problem as in classical LASSO, the ℓ_1-norm term enforces sparse inputs at the cost of possibly not reaching (or delaying) the desired state. Notice also that because we eliminated the state variable $\overline{x}$ from (2) to (5), the system dynamics are always (implicitly) satisfied.

Reduced precision (RP). Some platforms require representing arithmetic numbers with short binary codes, thereby reducing their accuracy. There are three main categories of representation: floating point [8], Q fixed point [9], and universal numbers (Unum) [10].

The IEEE 754 floating point arithmetic standard [11] represents a number by using three elements: a sign (1 bit), an exponent (n bits), and a mantissa (m bits). That is,

$$\text{sign} \times \text{mantissa} \times 2^{\text{exponent}}\,. \tag{7}$$

The Q fixed point format uses instead a fixed number of bits to represent the integer and the fractional parts of a number. Specifically, a number is represented by its sign (1 bit), its integer part (t bits), and its fractional part (k bits):

$$\text{sign} \times (2^{\text{integer}} + 2^{-\text{fraction}})\,. \tag{8}$$

Unum arose as an alternative to IEEE 754, and there are several versions. For example, Type III Unum, Posit, represents numbers by their sign (1 bits), regime (g bits), exponent (p bits), and fractional part (c bits):

$$\text{sign} \times (2^{2^p})^{\text{regime}} \times 2^{\text{exponent}} \times \left(1 + \frac{\text{fraction}}{2^c}\right). \tag{9}$$

Each of these representations has a different dynamic range. And different dynamic ranges affect not only the performance of the algorithm by limiting the type of operations that can be performed, but also their embedded implementation.

III. Algorithm and Implementation

We now explain the algorithm we use to solve LASSO MPC and its implementation on resource-constrained systems.

A. Proximal Gradient Descent

We consider the LASSO problem in (6) and apply the proximal gradient descent (PGD) algorithm with fixed step size [12]. PGD solves problems of the form

$$\underset{\overline{u}}{\text{minimize}} \;\; f(\overline{u}) := g(\overline{u}) + h(\overline{u})\,, \tag{10}$$

where $g : \mathbb{R}^q \to \mathbb{R}$ is convex, differentiable, and its gradient is Lipschitz-continuous, i.e., there exists $L > 0$ such that $\|\nabla g(\overline{u}) - \nabla g(v)\|_2 \leq L\|\overline{u} - v\|_2$, for all $\overline{u}, v$. The function $h : \mathbb{R}^q \to \mathbb{R} \cup \{+\infty\}$ is assumed convex and closed. Given a stepsize $\alpha \leq 1/L$ and an initial point $\overline{u}^0$, PGD solves (10) by iterating (on k)

$$\overline{u}^{k+1} = \text{prox}_{\alpha h}\big(\overline{u}^k - \alpha\nabla g(\overline{u}^k)\big)\,, \tag{11}$$

where the proximal operator of a convex, closed function ϕ at a point u is defined as

$$\text{prox}_\phi(\overline{u}) := \underset{v}{\arg\min} \;\; \phi(v) + \frac{1}{2}\|v - \overline{u}\|_2^2\,. \tag{12}$$

It is well known that the iterates produced by PGD satisfy [12]

$$f(\overline{u}^k) - f(\overline{u}^\star) \leq \frac{\alpha L\|\overline{u}^0 - \overline{u}^\star\|_2^2}{2k}\,, \tag{13}$$

where $\overline{u}^\star$ is a solution of (10), i.e., PGD converges sublinearly.

Application to LASSO MPC. As problem (6) has the format of (10) with $g(\overline{u}) = (1/2)\|H\overline{u} - y\|_2^2$ and $h(\overline{u}) = \lambda\|\overline{u}\|_1$,

the application of PGD is immediate and yields the soft-thresholding algorithm. Specifically, (11) becomes

$$\begin{aligned}\overline{u}^{k+1} &= \mathcal{S}_\lambda\Big(\overline{u}^k - \alpha H^\top (H\overline{u}^k - y)\Big) \\ &= \mathcal{S}_\lambda\Big(\big(I_q - \alpha H^\top H\big)\overline{u}^k + \alpha H^\top y\big)\Big)\,, \end{aligned} \tag{14}$$

where the soft-thresholding operator $\mathcal{S}_\lambda(\overline{u})$ applies to component i, for $i = 1, \ldots, q$, the following nonlinearity:

$$\Big[\mathcal{S}_\lambda(\overline{u})\Big]_i = \begin{cases} \overline{u}_i - \lambda & , \ \overline{u}_i > \lambda \\ 0 & , \ -\lambda \le \overline{u}_i \le \lambda \\ \overline{u}_i + \lambda & , \ \overline{u}_i < -\lambda\,. \end{cases} \tag{15}$$

Parameters and precomputations. PGD (11) and the associated convergence result in (13) apply whenever the stepsize α satisfies $\alpha \le 1/L$, where L is a Lipschitz constant of ∇g. To compute it, we can estimate the maximum eigenvalue of $H^\top H$, $\lambda_{\max}(H^\top H)$, e.g. by Lanczos's method, or use the structure of H and the matrices that define it:

Lemma 1. *Let $H = (\overline{B}^\top \overline{Q}\,\overline{B} + \overline{R})^{1/2}$, where $\overline{B}$, $\overline{Q}$, $\overline{R}$ are defined in* (4)-(5)*. Also, partition $\overline{B}$ vertically into $\overline{B}_1 \in \mathbb{R}^{nT \times mT}$, which contains the first nT rows of $\overline{B}$, and into $\overline{B}_2 \in \mathbb{R}^{n \times mT}$, which contains the last n rows of $\overline{B}$. Then,*

$$\begin{aligned}\lambda_{\max}(H^\top H) \le\ & \lambda_{\max}(Q)\lambda_{\max}\big(\overline{B}_1\overline{B}_1^\top\big) \\ & + \lambda_{\max}(P)\lambda_{\max}\big(\overline{B}_2\overline{B}_2^\top\big) + \lambda_{\max}(R)\,. \end{aligned} \tag{16}$$

The proof uses the subadditivity of $\lambda_{\max}(\cdot)$, and the properties of the Kronecker product; it is omitted for brevity. The matrices P, Q, and R encode the objective of MPC [cf. (3)], and the matrices $\overline{B}_1$ and $\overline{B}_2$ encode the system dynamics through their dependency on A and B [cf. (1)]. Although the matrices $\overline{B}_1$ and $\overline{B}_2$ have a lot of structure, namely $\overline{B}_1$ is block Toeplitz and $\overline{B}_2$ is the controlability matrix with permuted columns, it is not immediate to obtain a bound on $\lambda_{\max}\big(\overline{B}_1\overline{B}_1^\top\big)$ and $\lambda_{\max}\big(\overline{B}_2\overline{B}_2^\top\big)$ as a function of A and B. These quantities can be accurately estimated via Lanczos's method.

Once α is set to the inverse of right-hand side of (16), the matrix $I_q - \alpha H^\top H$ and the vector $H^\top y = \overline{B}^\top \overline{Q}\,\overline{A}\,x_0$ in (14) can be precomputed before the iterations of the algorithm. We stop the algorithm whenever a maximum number of iterations $k_{\max}$ is reached or when $|f(\overline{u}^{k+1}) - f(\overline{u}^k)| < \epsilon$, for some defined ϵ.

B. Approximate Core Synthesis

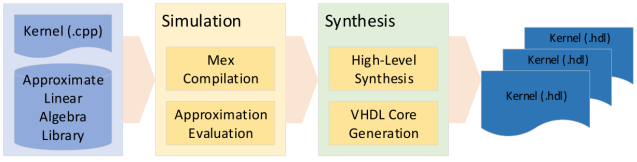


Fig. 1: Approximate kernel synthesis.

We now describe a proof-of-concept infrastructure that generates an approximate optimizer for (5) on a reconfigurable device. Fig. 1 shows the workflow of this infrastructure. The optimizer kernel is a user-defined C++ function based on an approximate linear algebra library, which defines basic algebra operations such as addition, multiplication, inversion, and decomposition, for matrices and vectors. We have developed that algebraic library for arbitrary precision representations.

We solved LASSO MPC using different precisions using an integrated environment with the Matlab MEX API. By compiling the MEX files with the proposed kernel, we can evaluate the functionality and algorithmic performance of the algorithm. After checking the correctness of the kernel via functionality simulation, the kernel is synthesized using high-level synthesis tools, and the approximate core is generated in VHDL.

IV. Experiments

We now describe experiments using LASSO MPC to control the attitude of a spacecraft [13] by ACADO [14]. Fig. 2 shows the state parameters for attitude control using reaction wheels. The state vector, corresponding to x in (1), is defined by $[roll, pitch, yaw, \omega_\omega, \omega_1, \omega_2, \omega_3]$ in Fig. 2. The control voltages for the reaction wheels that steer the spacecraft, $[\tau_1, \tau_2, \tau_3, \tau_\omega]$, are considered as the components of the input vector, u, in (1). In our experiments, the dynamic matrices A and B in (1) and the cost matrices P, Q, and R in (3) were set exactly as in [13].

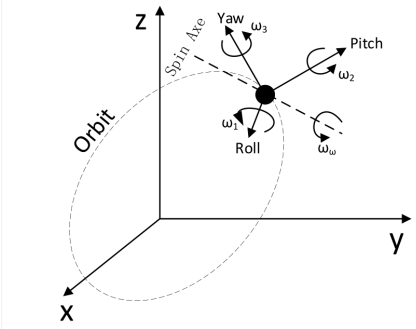
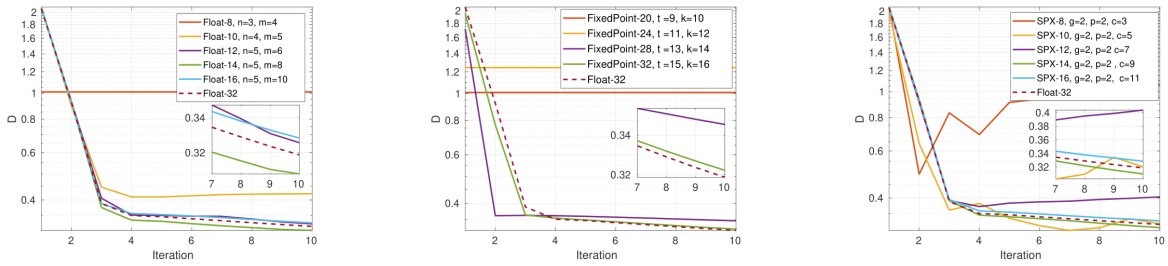


Fig. 2: Attitude Control [13]: seven states are considered here, $Roll, Pitch, Yaw, \omega_1, \omega_2, \omega_3, \omega_\omega$, where $Roll, Pitch, Yaw$ describe the rotating angles of the body frame relative to the orbit frame, $\omega_1, \omega_2, \omega_3$ are the corresponding angular velocities, and ω_ω is the angular velocity along the spin axis. The wheels are controlled by the input voltages, τ_1, τ_2, τ_3, τ_ω.

Experimental setup. We considered three different time horizons T: 1, 5, and 10. To assess our implementation of PGD, we compared its solution with the one returned by CVX [15], and used the sum of the absolute differences, $D = |u^k_{RP} - u_{cvx}|_1$ for k^{th} iteration, of the control input vector, $[\tau_1, \tau_2, \tau_3, \tau_\omega]$, as the performance metric. For example, $D^k_{Float-10} = |u^k_{Float-10} - u_{cvx}|_1$ is the solution between floating point using 10 bits and CVX at the k^{th} iteration, where $u^k_{\text{Float-10}}$ is the solution of (6) using a 10 bit floating point representation, and u_{cvx} is the solution returned by CVX.

A. RP Approximation Performance

We implemented the framework described in section III-B with different number representations, each with its own preci-



(a) FP (b) FXP (c) UP

Fig. 3: Convergence performance of approximate PGD (14) between the baseline 32-bit floating point (FP) and reduced precision (RP) implementations using (a) Floating Point (FP), (b) Fixed Point (FXP), and (c) Unum Posit (UP).

sion. Fig. 3a shows the baseline, single precision floating point (32 bits) simulation using CVX as the Lasso MPC solver, compared against our reduced precision implementations described in paragraphs (a), (b), and (c) below. Due to limited space, we only consider $T = 10$ here. Similar phenomena happen for $T = 1$ and $T = 5$.

(a) Floating Point (FP): In Fig. 3a RP floating point solutions are presented for PGD from 6-bit to 16-bit. As shown, performance with 8 to 10 bits is much worse compared to single precision float. However, from 12 bits the performance is considerable, which is similar to single precision floating. Hence, 12-bit floating point approximation is considered as a viable candidate for approximate FP implementation.

(b) Fixed Point (FXP): In Figure 3b, RP fixed point is presented for PGD from 20-bit to 32-bit. As shown, proper convergence does not occur for 20 or 24 bit approximation, while similar performance is obtained using 32 bits. Hence, 32-bit fixed point is considered as a candidate for approximate FXP implementation.

(c) Unum Posit (UP): In Fig. 3c, the RP unum posit is considered for PGD from 8 to 16 bits. Compared to the single precision float, similar performance is obtained from 14 bits which makes it the candidate of RP approximate PGD as well.

Fig. 4 shows the comparison of the the control state in an open-loop simulation with different RP techniques. As shown in Fig. 4b-4d, compared to the CVX solution using the baseline 32 point FP precision solution of Fig. 4a, the approximate solver shows very similar control process and performance to CVX over time. For example, in Fig. 4b, the PGD optimized input using 12 bits FP exhibits slightly larger variance on than CVX, while the PGD optimized output for Euler and Angular control takes about 1.5s to converge rather than about 1.1s with CVX. However, an approximation with over-reduced precision causes instability in the spacecraft attitude is shown in Fig. 4e.

B. Cost Evaluation

Choosing the closest MPC performance of each RP approximation to CVX across the different precisions shown in the last section, the corresponding costs are evaluated by implementing the PGD kernel on an Xilinx Ultrascale+ ZCU106 device as shown in Table I. The single 32-bit FP precision implementation is considered again as the baseline against which other representations are compared. All costs are estimated from high level synthesis using the Xilinx Vivado design tool 2019.2.

Across different horizon lengths, the approximate fixed point implementation (FXP-24,28,32) consumes the least logic area, which is about 30% of the baseline (FP32) in each case. Accordingly, the power consumption is reduced to $\tilde{2}5\%$ when the horizon length is 1, and $<$50% when the horizon length is 5 or 10. As the problem size increases with horizon length, the number of bits for fixed point arithmetic grows from 24 to 32 to maintain a performance similar to the single precision implementation. This is due to the fact that the dynamic range of linear algebra, numeric computations is larger as the size increases.

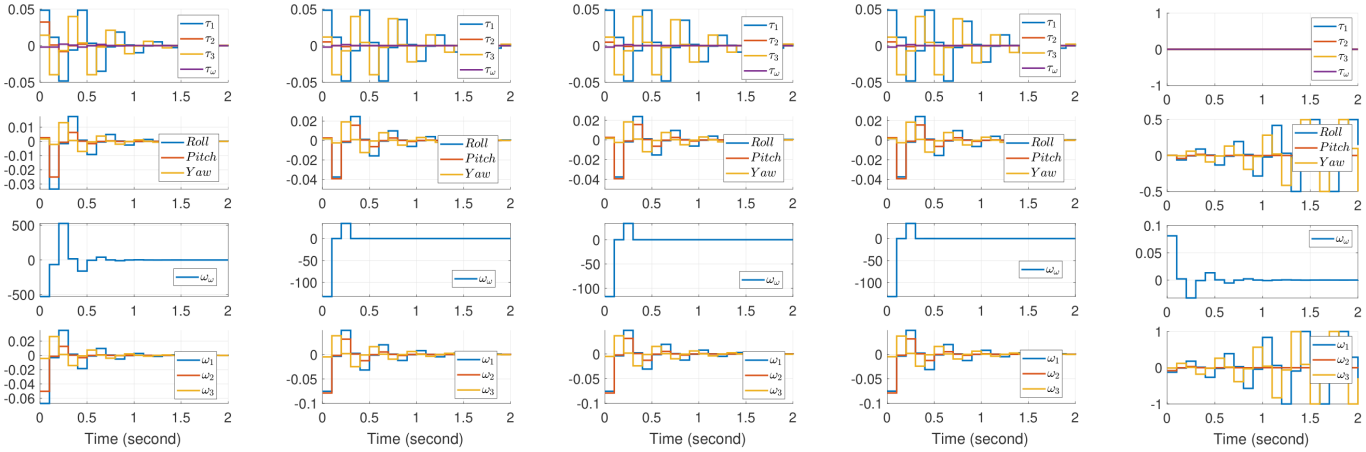
The (FP-12, 16,14) implementation does not save as much logic area as fixed point, but as the horizon length increases, up to 40% reduction in the use of LUT is achieved. Accordingly, up to $\tilde{2}0\%$ power reduction is introduced, but only within a very limited time horizon, $T - 1$. However, the least number of bits are adopted compared to fixed point implementations, enabling significant saving in communication bandwidth to 14.06% of FP-32.

The approximate unum posit (UP-12,14) is based on high level synthesis of SoftPosit [16]. It consumes significantly larger logic area comparing to floating point and fixed point implementations, although similar savings on communication bandwidth are made due to the low number of representative bits.

Hence, if power and area are of most concern on a resource constrained system, the fixed point (FXP) lean PGD does provide significant savings compared to the full, single precision, FP-32 implemnetation. However, if communication is of the most concern in a resource constrained system, such as a mesh network with edge devices, the FP floating point variations show the largest bandwidth savings.

V. Conclusions

In this work, an approximate proximal gradient descent is applied to solve Lasso MPC with fixed step size. By adopting the reduced precision technique, a considerable optimization performance is achieved compared to high-precision solver



(a) CVX (FP-32) (b) FP-12 (c) FXP-24 (d) UP-14 (e) FP-8

Fig. 4: LASSO MPC Simulation: $[\tau_1, \tau_2, \tau_3, \tau_\omega]$ is the control (input) vector in $Volt$ while $[roll, pitch, yaw, \omega_\omega, \omega_1, \omega_2, \omega_3]$ is the state (output) vector. $roll, pitch, yaw$ are measured in $Degrees$ and $\omega_\omega, \omega_1, \omega_2, \omega_3$ are measured in Rad/s. The simulation shows all input and output states changing over 2s with a 0.1s step interval.

TABLE I: RP Cost Comparison with Different Horizon Length (T)

	T=1				T=5				T=10			
Precision	FP-32	FP-12	FXP-24	UP-12	FP-32	FP-16	FXP-28	UP-14	FP-32	FP-14	FXP-32	UP-14
LUT ($\times 10^3$)	3.31	2.99	1.21	19.4	3.17	2.16	1.24	10.9	4.01	2.42	1.46	10.6
DSP48E1	30	0	11	12	20	6	10	4	20	6	16	4
BRAM	0	0	0	0	2	2	4	0	5	5	5	8
Clock (MHz)	482	465	443	393	434	401	403	382	370	384	379	382
T (M Inst/sec)	0.42	0.48	0.46	0.41	0.45	0.41	0.42	0.39	0.38	0.39	0.39	0.39
Power (mW)	273	199	68	248	219	220	110	152	250	254	113	148
Bandwidth	100%	14.06%	56.25%	14.06%	100%	25%	76.56%	19.14%	100%	19.14%	100%	19.14%

(CVX). An approximate core synthesis infrastructure is developed for fast prototyping the computational kernel of proximal gradient descent on reconfigurable device, FPGA. The results show up to 60% in logic cost reduction, 80% in memory bandwidth saving, and 70% in power reduction, which is very promising for deploying LASSO MPC on resource constrained system considering computing and communication cost. The future work includes further exploring the approximation effects against different MPC applications as well as different optimization algorithms.

REFERENCES

[1] S. Qin and T. A. Badgwell, "A Survey of Industrial Model Predictive Control Technology," Control Engineering Practice, vol. 11, no. 7, pp. 733 – 764, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066102001867

[2] M. Gallieri and J. M. Maciejowski, "LASSO MPC: Smart Regulation of Over-Actuated Systems," in 2012 American Control Conference (ACC), 2012, pp. 1217–1222.

[3] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in 2016 IEEE International Conference on Rebooting Computing (ICRC), 2016, pp. 1–8.

[4] K. Ling, B.-F. Wu, and J. Maciejowski, "Embedded Model Predictive Control (MPC) using a FPGA," IFAC Proceedings Volumes (IFAC-PapersOnline), vol. 17, 01 2008.

[5] A. Beck and M. Teboulle, "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems," SIAM J. Im. Sc., vol. 2, no. 1, pp. 183–202, 2009.

[6] F. Borrelli, A. Bemporad, and M. Morari, Predictive Control For Linear And Hybrid Systems. Cambridge University Press, 2017.

[7] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," J. Royal Statistical. Soc., Series B, vol. 58, no. 1, pp. 267–288, 1996.

[8] G. Flegar, F. Scheidegger, V. Novaković, G. Mariani, A. E. Tom´ s, A. C. I. Malossi, and E. S. Quintana-Ortí, "FloatX: A C++ Library for Customized Floating-Point Arithmetic," ACM Transactions on Mathematical Software (TOMS), vol. 45, no. 4, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3368086

[9] S. W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing. USA: California Technical Publishing, 1997.

[10] J. Gustafson and I. Yonemoto, "Beating Floating Point at Its Own Game: Posit Arithmetic," Supercomputing Frontiers and Innovations, vol. 4, pp. 71–86, 01 2017.

[11] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefvre, G. Melquiond, N. Revol, and S. Torres, Handbook of Floating-Point Arithmetic, 2nd ed. Birkhäuser Basel, 2018.

[12] A. Beck and M. Teboulle, Convex Optimization in Signal Processing and Communications. Cambridge University Press, 2010, ch. Gradient-based algorithms with applications to signal-recovery problems.

[13] Øyvind Hegrenæs, J. T. Gravdahl, and P. Tøndel, "Spacecraft Attitude Control using Explicit Model Predictive Control," Automatica, vol. 41, no. 12, pp. 2107 – 2114, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109805002657

[14] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," Optimal Control Applications and Methods, vol. 32, no. 3, pp. 298–312, 2011.

[15] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[16] S. H. Leong, NGA SoftPosit, 2017 (accessed July 3, 2019). [Online]. Available: https://gitlab.com/cerlane/SoftPosit