An Efficient Accelerator for Nonlinear Model Predictive Control

Sergio A. Pertuz*†, Ariel Podlubne*†, Diana Goehringer*†
*Chair of Adaptive Dynamic Systems, Technische Universität Dresden, Germany
†Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, Germany
{sergio.pertuz, ariel.podlubne, diana.goehringer}@tu-dresden.de

***Abstract*—The computational complexity of Nonlinear Model Predictive Control (NMPC) often hinders their application to cyber-physical systems with fast dynamics, such as mobile robots or Unmanned Aerial Vehicles. This complexity overhead comes from the control algorithm's backbone, an iterative solver that must ensure convergence and often takes the form of a highly structured convex Quadratic Program (QP). Such overhead could be overcome using specialized computer architectures. Field Programmable Gate Arrays are good candidates for making hardware accelerators that comply with the real-time constraints of fast-dynamic cyber-physical systems. Nevertheless, QP-solvers have been demonstrated to be complex to implement as a hardware accelerator. With this in mind, the present paper proposes a novel accelerator architecture that uses Knowledge-based Particle Swarm Optimization (PSO) as a solver while exploring its parallel nature. PSO is a stochastic global optimization algorithm that creates a fast and precise solution for NMPC. The proposed strategy in this papergrants system control stability for short sampling frequencies and long prediction horizons. It can also meet real-time constraints while achieving low hardware consumption. Additionally, it is generalized, so it can potentially be adapted to any application and is compatible with the Robot Operating System (ROS). The architecture is tested with two applications: an inverted pendulum swing-up procedure and a quadrotor drone with control and state constraints. Following, we analyze the accelerator performance and highlight our solution's advantages to other works in the literature. Namely, our architecture solves more complex problems with a greater dimension and longer horizon while using similar resources. The proposed solution also has good computational performance (29ms and 11ms) for both the quadrotor and inverted pendulum, respectively, while achieving the real-time requirements (50ms and 100ms, respectively). Parallelly, ad-hoc embedded architectures are important for a low-end, low-cost, and low-power MPSoC+FPGA device. Our solution uses less than 50% of a low-end, low-power MPSoC device (ZU3EG), while others rely on large, more power-hungry devices (e.g., Kintex7 and XC7Z045).**

***Index Terms*—Memory-based particle swarm optimization, HW/SW co-design, nonlinear model predictive control, robotics,**

I. Introduction

Research and application of Model Predictive Control (MPC) have steadily grown in the last decade [1]. Its ability to directly embed hard constraints in the control law and provide performance closer to an optimal control attracts its use. However, since the control action in MPC is computed by solving an optimization problem at each control sampling time, these features come at a high computational cost. The reason for this increased computational cost is because optimization algorithms need for repeated computations and evaluations of mathematical functions to search for the optimal solution. This is especially true when dealing with nonlinear systems where the complexity of the underlying models and datasets involved in increased.

Nonlinear MPC (NMPC) treats nonlinear dynamics and constraints directly and explicitly without approximation [2]. In this case, the cost function is non-convex and requires solving a nonlinear optimization problem at each control sampling time with a higher computational cost.

Applying NMPC to fast dynamic systems, such as automotive, aerospace, and robotic applications is still challenging. Moreover, in cases where the controller has to run on an embedded system with low power constraints, NMPC becomes an even more challenging problem to solve. Even though embedded processors have significantly advanced in the last decade due to the mobile devices industry, their general-purpose architecture for running software and its power constraints still limit their frequency and performance. With this in mind, developing an optimized accelerator on reconfigurable hardware platforms, such as Field-Programmable Gate Arrays (FPGAs), becomes an interesting choice. In this case, the reason is that FPGA can be used to explore parallelism to speed up algorithm performance and decrease power consumption.

In this context, this work presents as its main contributions:

a a custom-designed hardware accelerator based on High Level Synthesis (HLS) with a framework that allows its application to different nonlinear systems with long horizons and
b an evaluation of a relatively low-power and small FPGA device with two different systems: a swing-up procedure on a constrained inverted pendulum and a quadrotor drone model.

The remainder of this paper is organized as follows. Sections II and III provide brief theoretical overviews of the NMPC and PSO algorithms, followed by the related works. Then, Section IV presents the enhancements, and the proposal of the PSO adapted to solve the NMPC problem efficiently. Section V presents the implementation of the proposed controller algorithm as a ROS-compatible FPGA accelerator and its parallel strategies. Section VI with the evaluation of both the KPSO-NMPC, as mentioned above, using the models of our case study and the accelerated architecture's performance. Finally, Section VI discusses the results, while the conclusions and future works finish the document in Section VII.

DOI 10.1109/ASAP57973.2023.00038

II. Background

A. Nonlinear model predictive control

NMPC uses a system's model to predict its behavior over a finite horizon and compute an optimal control action to follow a given reference. The nonlinear system can be described in a discrete-time state-space model by Eq. (1) and Eq. (2).

$$\mathbf{x}^{(t+1)} = f(\mathbf{x}^{(t)}, \mathbf{u}^{(t)}), \quad t \geq 0 \tag{1}$$

$$\mathbf{y}^{(t)} = h(\mathbf{x}^{(t+1)}) \tag{2}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, $\mathbf{x}^{(t)} \in \mathbb{R}^n$ is the state vector, $\mathbf{u}^{(t)} \in \mathbb{R}^m$ is the system input vector, $\mathbf{y}^{(t)} \in \mathbb{R}^n$ represents the controlled output vector, and $h : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the function that maps the state vector to the controlled outputs.

Based on the system model, the NMPC optimization problem can be defined as the minimization of a cost function (J) to obtain the optimal control sequence as follows:

$$\min J(\mathbf{x}^{(t)}, \mathbf{u}^{(t)}) \triangleq \overbrace{\sum_{i=1}^{N-1} \left\| \hat{\mathbf{x}}^{(t+i)} - \mathbf{r}^{(t+i)} \right\|_{\mathbf{Q}}^2}^{\text{Error State}} + \underbrace{\left\| \hat{\mathbf{x}}^{(N)} - \mathbf{x}_{ss} \right\|_{\mathbf{Qf}}^2}_{\text{Error Final Estate}} + \underbrace{\sum_{i=0}^{Nc-1} \left\| \hat{\mathbf{u}}^{(t)} - \mathbf{u}_{ss} \right\|_{\mathbf{R}}^2}_{\text{Error Control}} \tag{3}$$

where $\hat{\mathbf{x}}^{(t)}$ and $\hat{\mathbf{u}}^{(t)}$ are the solver's estimated system state vectors and control action inputs, respectively. $\mathbf{r}^{(t)}$ is the system reference vector at state t, and $\mathbf{x}_{ss}$ and $\mathbf{u}_{ss}$ are the vectors steady-state output and control values. N is the prediction horizon, Nc is the control horizon ($Nc \leq N$), and **Q**, **Qf**, and **R** are the diagonal weighting matrices for the intermediate state, steady-state, and control input deviation, respectively.

The system is also subject to constraints on input ($\mathbf{u}_{min}$, $\mathbf{u}_{max}$), input variation ($\Delta\mathbf{u}_{min}$, $\Delta\mathbf{u}_{max}$), and controlled outputs ($\mathbf{y}_{min}$, $\mathbf{y}_{max}$).

In summary, the NMPC scheme follows three steps which are repeated at each sampling time: (a) Measure the system output $\mathbf{y}^{(t)}$ and compute/estimate its state $\mathbf{x}^{(t)}$; (b) Compute $\mathbf{u}_{opt}^{(t)}$, the optimal control sequence, by solving the finite horizon optimal control problem; (c) Apply the first optimal control action $\mathbf{u}^{opt}(1)$ at instant k. This work uses the PSO to solve the nonlinear optimization problem and is introduced in the next section.

B. Particle Swarm Optimization

The PSO is a nature-inspired algorithm of the swarm intelligence field that uses particles to search in an N-dimensional space $\mathbb{G}$ to find the maximum or minimum value of a given function [3]. Each particle has its position $u_i \in \mathbb{G} \in \mathbb{R}^n$ and velocity $v_i \in \mathbb{R}^n$ at a given iteration.

The canonical algorithm initializes each particle with uniformly distributed random values.At each iteration, the particle's position is evaluated by a cost function and stored in an individual best variable (y_i) if the evaluation is improved. Each best is compared among the others and against the global best position (y_{best}). This y_{best} is updated at each iteration. Afterward, each particle's velocity and position are updated as described in Eqs. (4) and (5)

$$v_{i,j}^{(k+1)} = w v_{i,j}^{(k)} + c_1 \overbrace{U_1(y_{i,j}^{(k)} - u_{i,j}^{(k)})}^{\text{cognitive component}} + c_2 \overbrace{U_2(y_{sj}^{(k)} - u_{i,j}^{(k)})}^{\text{social component}} \tag{4}$$

$$u_{i,j}^{(k+1)} = u_{i,j}^{(k)} + v_{i,j}^{(k+1)} \tag{5}$$

Where $k \in (1,\ k_{max})$ represents the current iteration, $i \in (1,\ S)$ is the particle index, and $j \in (1,\ \text{Dim})$ is the index for the particle's dimension. w is the inertia factor, which starts with a high value helping the particles explore the search space faster and then decays linearly with each iteration. U_1 and U_2 are random numbers in the interval $[0, 1]$. v and u are limited by $[-v_{max},\ v_{max}]$ and $[u_{min},\ u_{max}]$. c_1 is the cognitive coefficient (confidence in the particle's best), and c_2 is the social coefficient (confidence in the global best).

III. Related Work

The main two compute-intensive design choices in accelerating NMPC are the control horizon and the nonlinear optimization solver. Previous efforts of the controller's FPGA implementations in the literature [4]–[7] use HW/SW Co-design and achieve a reasonable execution time. However, because they use some variant of the Quadratic Programming (QP) gradient-based method, the resulting architectures rely on large FPGAs and have small prediction horizons. A faster approach is presented in [8], a modified version of the continuation generalized minimal residual method. However, because they involve solving a set of serial quadratic equations subject to linear constraints, QP-solvers are not well suited for hardware acceleration on FPGAs, which are optimized for parallelism and pipelining. .

Other solutions in the literature implement Deep Neural Networks (DNN) as a linear solver [9], where the DNN computes the control law and adopts a simple integer module to compensate for the approximation error. These solutions can be high-speed (0.12ms execution time for drone control) and compact. However, they require the architecture to be retrained every time the controlled system changes (i.e., for a new application).

Another category of nonlinear solvers are stochastic global optimization algorithms such as evolutionary algorithms and swarm intelligence-based ones [10]. These solvers stand out for their relative simplicity and highly parallel nature, which makes them very suitable for fast hardware implementation.

PSO solvers in NMPC are found in recent works [11], [12] and present fast PSO implementations on FPGA solutions in fixed-point arithmetic. These variants have one to two input

variables, one controlled output, and short horizons. [13] also propose some modifications to the PSO algorithm with a more stable and faster solution.

Additionally, most previous works do not show effort in generalizing the controller architecture. This inexisting generalization also happens in [14], where the accelerated MPC is focused only on mobile robot applications. On the other hand, [15] creates a framework for FPGA-based accelerated NMPC that uses grid-based MPC that achieves low latency and goes from the problem definition through FPGA accelerator deployment.

Finally, all of the works mentioned above do not consider the software components that robotic applications may have. They also do not consider how the controller can be integrated into existing platforms and applications. They also do not show support for any robotics middleware such as the Robotic Operative System (ROS).

IV. PSO ENHANCEMENTS FOR NMPC

Beyond adjusting the PSO constant parameters to suit the problem, this work proposes enhancements for better efficiency and NMPC compliance. In the NMPC problem, the control actions' search space is limited, and this limitation is embedded in PSO. However, the control input variation from one sampling time to another ($\Delta\mathbf{u}_{max}$) is not limited by PSO. Furthermore, another limitation is added when updating and initializing particles to deal with these variation constraints.

A variation for the PSO is described in [16], where the algorithm uses available knowledge about a specific problem. In this work, this variation is adapted to NMPC where the particles are initialized with the solution from the previous NMPC run. In this case, this solution containing Nc control actions is shifted by one sampling period, discarding the first control. The variation is further enhanced by initializing one of the particles with all dimensions equal to $\mathbf{u}_{ss}$ (control value at steady state $\mathbf{x}_{ss}$). Similarly, another particle is initialized with a velocity equal to zero and the local best $\mathbf{y} = \mathbf{x}_{ss}$. This procedure improves the control signal noise and maintains system stability at a steady state.

Considering all the previous modifications, the overall NMPC strategy is described in Algorithm 1.

A. Cost Function

The cost function (see Eq. (3)) includes a predictor that adopts a fourth-order Runge-Kutta (RK4) method [17], using four weighted predictions of the system using the model at each time step, obtaining a more accurate prediction. This method is included as one of the main parts of the cost function, which are three: 1) the error adder, 2) the RK4 algorithm, and 3) the system model. Furthermore, the cost function can be generalized for different controlled systems leaving the last two modules and only changing the first. In this way, the system model, which implements the differential equations, can be exchanged depending on the application and the constraints. For the *Error State* and *Error Final State*, it is necessary to predict the system's states with the horizon inputs given by each particle ($\hat{y}$).

Algorithm 1 Pseudo-code for the NMPC

1: **function** NMPC($\mathbf{x}^{(t)}$, $\mathbf{r}^{(t)}$, $\mathbf{y}_{best}^{(t-1)}$)
2: Get measured states from $\mathbf{x}^{(t)}$ and reference trajectory $\mathbf{r}^{(t)}$ into $J(\cdot)$
3: Randomly initialize particles obeying $\mathbf{u}_{min}$, $\mathbf{u}_{max}$, $\Delta\mathbf{u}_{min}$, $\Delta\mathbf{u}_{max}$
4: Initialize one particle with global best from previous solution ($\mathbf{y}_{best}^{(t-1)}$)
5: Initialize one particle with all dimensions equal to $\mathbf{u}_{ss}$
6: Get $\mathbf{u}_{ctrl}^{(t-1)}$ from the first horizon step of $\mathbf{y}_{best}^{(t-1)}$
7: **repeat**
8:
9: **for** i **do** 1 to S
10: **if** $J(u_i^{(k-1)}) \leq J(y_i^{(k)})$ **then**
11: $y_i^{(k)} \leftarrow u_i^{(k-1)}$
12: **end if**
13: **end for**
14:
15: get y_{best} fitness values from $J(y^{(k)})$
16:
17: **for** i **do** 1 to S
18: $u_{i,0} \leftarrow \mathbf{u}_{ctrl}^{(t-1)}$
19: **for** j **do** 1 to N
20: $v_{i,j}^{(k+1)} \leftarrow wv_{i,j}^{(k)} + c_1U_1(y_{i,j}^{(k)} - u_{i,j}^{(k)}) + c_2U_2(y_b^{(k)} - u_{i,j}^{(k)})$
21: $u_{i,j}^{(k+1)} \leftarrow u_{i,j}^{(k)} + v_{ij}^{(k+1)}$
22: $\Delta u_j^{(k)} \leftarrow u_{i,j}^{(k+1)} - u_{i,j-1}^{(k+1)}$
23: **if not**$(\Delta u_j^{(k)} \in [\Delta\mathbf{u}_{min}, \Delta\mathbf{u}_{max}])$ **then**
24: $u_{i,j}^{(k+1)} \leftarrow u_{i,j}^{(k)} \pm \Delta\mathbf{u}_{max}$
25: **end if**
26: **end for**
27: **end for**
28:
29: Increment k
30:
31: **until** $(J(y_{best}) < J_{tresh})$ or $(k > k_{max})$
32: Apply first control action to the system $\mathbf{u}_{ctrl}^{(t)} \leftarrow {y_{best}}_1$
33: **return** The best fit particle y_{best} and its fitness $J(y_{best})$
34: **end function**

Initialize Particles
Evaluation and detection of the best individual
Global best
For init $\Delta\mathbf{u}_{min/max}$ constraints
Constraining with $\Delta\mathbf{u}_{min/max}$
Update

V. CO-PROCESSING CORE IMPLEMENTATION

A. Optimized Parallelized Hardware Architecture

The PSO algorithm has easily exploitable characteristics for FPGA implementation, which is also true for the proposed NMPC. Algorithm 1 describes the NMPC and, in it, the main loop (on Line 7) represents each iteration. The inner loops that compose the *update* and *evaluation and detection of best individual* (on Lines 9 and 17) can be executed in parallel as these loops are not dependant on previous iterations.

On the other hand, the loop on Line 19 is where the particles' position and velocity are updated, is iteration-dependant. This dependency is because each dimension represents a control action that obeys the $\Delta\mathbf{u}$ constraints (See *constraining with* $\Delta\boldsymbol{u}_{min/max}$ bracket). However, despite this dependency, each particle can be updated in parallel. Finally, The *global best* bracket, which compares the local best of each particle, is naturally a parallel procedure, provided all values can be organized in a comparison tree.

In the *evaluation and detection of best individual* block, the cost function ($J(\cdot)$) does the following steps:

1) fetch the particle's control horizon u_i^{k+1}, of the current i-particle and $(k+1)$-time step, from *update particles* block and broadcast it with *u guess* block,
2) computes the system states over the prediction horizon (N),
3) calculates the *error state*,
4) *error final state* and

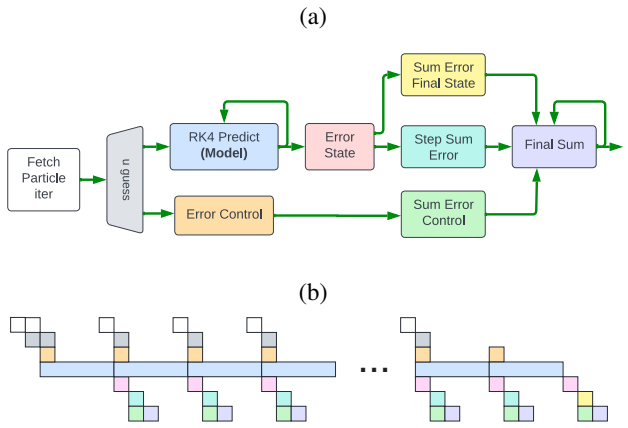

Fig. 1. Cost function's (a) dataflow and (b) pipelined stages. Blocks are color coded.

5) *error control* in Eq. (3),
6) sums each error separately, and
7) makes the final sum.

All the previous components can be executed in a coordinated dataflow manner, as shown in Fig. 1a.

Another essential thing to mention about the *cost function* is that it does not need to wait for the *update* loop to finish to start computing. Instead, every time a new $u_{i,j}^{(k+1)}$ is generated, it can execute one step horizon and perform all the steps mentioned for one dimension, thus merging and pipelining both the *update* and *evaluation and detection of the best indivitual* loops (see Fig. 1b). This novel dataflow-pipelined approach for the NMPC not only makes processing faster but also saves memory since only the current predicted state ($\hat{\mathbf{x}}^{(k)}$) has to remain in memory.

B. Block Design Hardware Architecture

The NMPC should be flexible enough to be applied to different system models with a configurable number of inputs, outputs, prediction, and control horizons, among all the other parameters listed in Table I. To that end, an architectural decision was made to use half-precision floating-point arithmetic for the model computation providing better precision and a broader dynamic range for models whose limits might not be easy to determine at design time. A fixed-point representation is used for all other hardware components since they involve only the particles' values and error calculation. A preliminary analysis showed that 32-bit (16-bit integer) provides a good compromise between hardware utilization and precision.

The hardware schematics of the NMPC accelerator are depicted in Fig. 2 where the blocks in blue are developed in HLS. *fit eval & local min* module (short for *evaluation and detection of the best individual*) represents the highest computational cost. It is composed of parallelized $CostF_i$ sub-modules that integrate the RK4 predictor (see Section IV-A) and a system model module (see Fig. 1a). The system model module implements the differential equations. These $CostF_i$

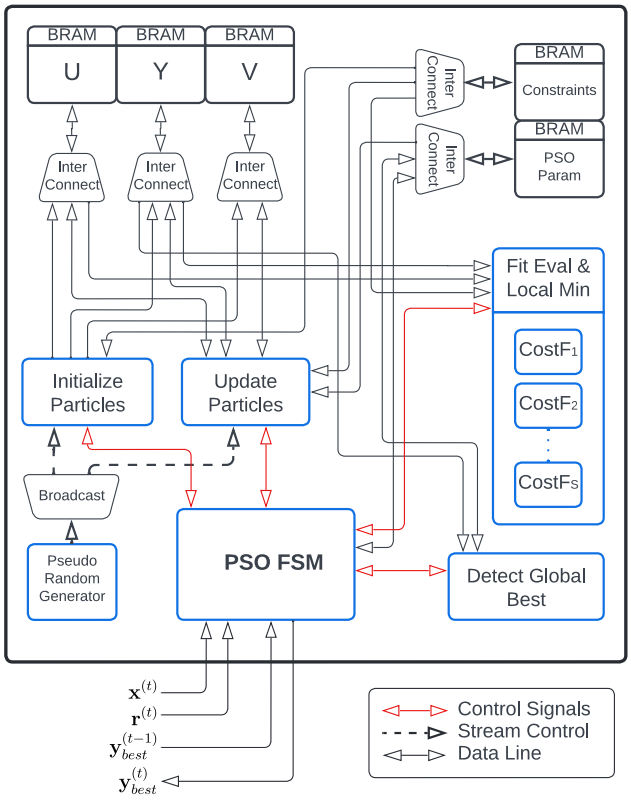

Fig. 2. NMPC Accelerator Hardware Architecture. All data lines are 32-bit-width.

modules are implemented in parallel, calculate the particles state error, detect the local minimum and then update the same memory. This module also limits the usage of multipliers and adders to reduce FPGA resource utilization by leveraging the HLS pragma `allocation` .

The other main sub-modules of the NMPC algorithm represent each PSO step, namely, *initialize particles*, *update particles*, and *detect global best*. As previously mentioned, these blocks are parallelized and pipelined for faster processing. Other necessary auxiliary blocks are the *pseudo random generator* which is designed in a pipelined way that allows it to generate a pseudo-random number every clock cycle on the interval $[0, 1]$ when enabled. The generator is based on the linear congruential generator algorithm [18].

To reduce register usage, the PSO particle variables (u, y, v, *system constraints* and *PSO paramateres* are stored in internal FPGA RAM blocks (BRAM or LUT depending on their size). The BRAM are 1WNR (1 write N reads, where $N = S$) memory blocks, and each memory address for all blocks stores 32-bit fixed-point variables for u, y, v represent one dimension of each particle. One write and N read is sufficient to increase the memory throughput for each loop because each dimension of all particles can be accessed simultaneously for parallel processing.

All these sub-modules are coordinated by an *KPSO FSM*

responsible for implementing the pipeline strategy between update and cost function depicted in Fig. 1.

C. Accelerator Interfacing and Customization

The NMPC architecture can be used in conjunction with a processor with an interface wrapper based on middleware specifications, such as ROS1/2, to ease their integration. For this interfacing, a tool [19] is used to automate the accelerator interface capable of quickly establishing incoming and outgoing communication from/to existing robotics systems.

The NMPC accelerator can be configured for any system, as long as the user provides the customization parameters on Table I and the system state space equations in C++ code respecting the function prototype:

Listing 1. Model Function Prototype

```
template <
 typename _model_real
>void __model(
 _model_real *state_dot,
 _model_real *state,
 _model_real *control
);
```

The accelerator is template-based and allows the user to change the model and some structural customization parameters. These structural parameters determine the memory size and the number of instances, such as the prediction and control horizons, the number of particles, the number of states, and control actions of each system are all configured by these parameters.

D. ROS integration

As mentioned, this work also introduces a ROS node that runs the KPAO-NMPC accelerator and is compatible with two target SoC: i) Zynq 7000 SoC™, and Zynq UltraScale+ MPSoC™ . The computations of the NMPC accelerator are naturally performed on the FPGA of the SoC. For testing purposes, the source code of the NMPC can also be run on the Arm®Cortex®CPU of the possible target SoC.

The NMPC ROS node subscribes to:

i the current measured state from the sensors ($\mathbf{x}^{(t)}$),
ii the current reference trajectory ($\mathbf{r}^{(t)}$), and
iii the previous system predicted control horizon ($\mathbf{y}_{best}^{(t-1)}$ respectively and as seen in Figs. 2 and 3)

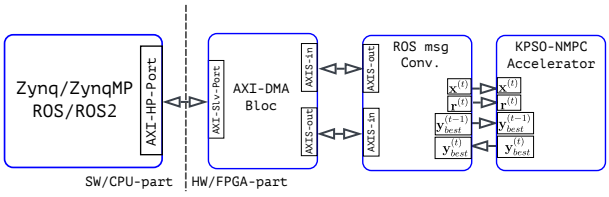

Fig. 3. ROS-based HW/SW accelerator.

The node then publishes the outcoming estimated control horizon (y_best as seen in Figs. 2 and 3).

The communication between the SoC CPUs and the FPGA leverages AXI-DMA and a module that serializes/deserializes the transformed messages, as seen in Fig. 3.

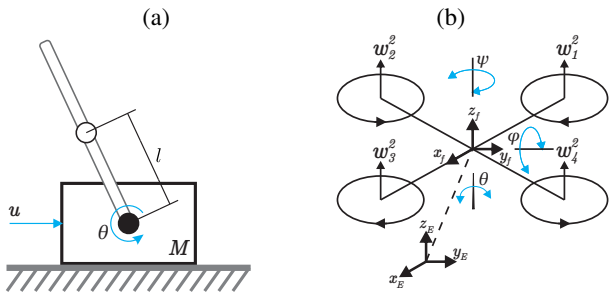

Fig. 4. Use case models: (a) inverted pendulum and (b) quadrotor drone.

VI. Evaluation

A. System Models

The case studies for the NMPC are an inverted pendulum system on a cart and a quadrotor drone. The inverted pendulum is described in Fig. 4a. It details the pendulum system whose model is non-linear and are taken from [20]. The states of the inverted pendulum are described by Eqs. (6) and (7)

$$\mathbf{x} = \left[x, \theta, \dot{x}, \dot{\theta}\right] \tag{6}$$

$$\mathbf{u}_{ctrl} = [F] \tag{7}$$

$[x, \theta]$ represent the position of the cart and angular position of the pendulum, and $\left[\hat{x}, \hat{\theta}\right]$ their respective speeds. Finally, the input u is a force (in Newtons) that controls the car.

The quadrotor drone has four rotors directed upwards from the center of mass of the quadrotor. The mathematical model for the quadrotor dynamics is derived from Euler-Lagrange equations [21].

The twelve states for the quadrotor are

$$\mathbf{x} = \left[x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}\right] \tag{8}$$

$$\mathbf{u}_{ctrl} = \left[w_1^2, w_2^2, w_3^2, w_4^2\right] \tag{9}$$

$$= \text{mma}\left([\text{Throttle}, \text{Roll}, \text{Pitch}, \text{Yaw}]\right) \tag{10}$$

where $[x, y, z]$ denote the spatial position of the quadrotor frame, $[\phi, \theta, \psi]$ are in the angle positions (roll, pitch, and yaw), and the remaining states are the derivatives of the previously mentioned states. The control inputs for the quadrotor are the squared angular velocities of the four rotors reinterpreted as the command thrust, roll, pitch, and yaw using a *motor mixing algorithm* (MMA).

B. Algorithm Evaluation

A software simulation of the controller is developed to evaluate the NMPC and executed several times to ensure it can solve the NPMC problem correctly.

Table I describes the parameters used for these experiments. The number of iterations and particles (k_{max}, and S respectively) is the smallest value still capable of controlling the system. These values were obtained experimentally and significantly impacted computational performance. The cognitive and social coefficients are set to $c_1 \geq c_2$ because the

TABLE I
PARAMETERS OF NMPC CONTROLLER. ✖ = IRRELEVANT.

Parameter	Inverted Pendulum	Quadrotor Drone
S	10	5
Dim	1	4
u_{min}	$[-50]$	$[-100, -100, -100, -100]$
u_{max}	$[50]$	$[100, 100, 100, 100]$
k_{max}	100	25
c_1, c_2		$[2.1, 1.0]$
ω_0, ω_f		$[0.9, 0.1]$
$\Delta\mathbf{u}$	$[50]$	$[20, 20, 20, 20]$
T_s	0.1	0.05
x_{min}	$[-2, ✖, ✖, ✖]$	$[✖, 0, ✖, -\pi/4, -\pi/4, ✖, ✖, ✖, ✖, ✖, ✖, ✖]$
x_{max}	$[2, ✖, ✖, ✖]$	$[✖, 0, ✖, \pi/4, \pi/4, ✖, ✖, ✖, ✖, ✖, ✖, ✖]$
$\mathbf{u}_{ss}$	$[0]$	$[0, 0, 0, 0]$
Q	$[2, ✖, 1, ✖]$	$[1.0, 1.0, 1.0, 3.0, 3.0, 3.0, ✖, ✖, ✖, ✖, ✖, ✖]$
Q_f	$[0.1, 0.015, 0.1, ✖]$	$[10, 10, 10, 30, 30, 30, ✖, ✖, ✖, ✖, ✖, ✖]$
R	$[0]$	$[5e{-}4, 5e{-}4, 5e{-}4, 5e{-}4]$

NMPC problem has multiple local minima with a multi-modal search space. For this type of problem, a larger c_1 is more advantageous given that it causes each particle to be attracted to its individual best position, resulting in a more sparse search [22].

Other parameters in the table are specific to the system and provided by the controller designer, except for $\Delta\mathbf{u}$. The latter affects the system's stability during its operation and is limited by its physical constraints. Some values are presented as irrelevant (✖) because they are either not controlled states or measured in degrees (whose values are cyclical).

Fig. 5 presents a comparison for the use-cases NMPC control using three solver methods: 1) an Active-Set Quadratic-Programming (QP) as optimal solution [23], 2) a Canonical PSO, and 3) the proposed NMPC.

Fig. 5 shows struggles of the Canonical PSO to find adequate control solutions in each time iteration and that the system inputs have a considerable variance. On the other hand, the NMPC algorithm can find a stable control for the system reaching the desired reference faster and reducing control action variation after the system reaches the desired position. Finally, it is also possible to determine from Fig. 5 that the proposed NMPC approach can reach a similar solution to QP, considered the optimal solution.

Table II describes the MSE performance of the solvers, and it is split into two parts *Error State*, and *Error Control* and are the mean of those values for each time-iteration (see Equation (3)). It further proves the overall improvement of NMPC over canonical PSO and proximity to QP-solution. It is also relevant to mention that, when given a large number of particles and iterations, the Canonical PSO can achieve similar results to the *PSO; however, this results in a more considerable computational effort.

TABLE II
MEDIUM SQUARED ERROR (MSE) OF NMPC IMPLEMENTATIONS. *PSO IS PSO WITH VARIATIONS

	Error State			Error Control		
System	QP	PSO	*PSO	QP	PSO	*PSO
Inv. Pend.	0.02	0.05	0.02	190.59	378.43	223.27
Drone	0.10	0.39	0.06	9.00	2316.95	268.22

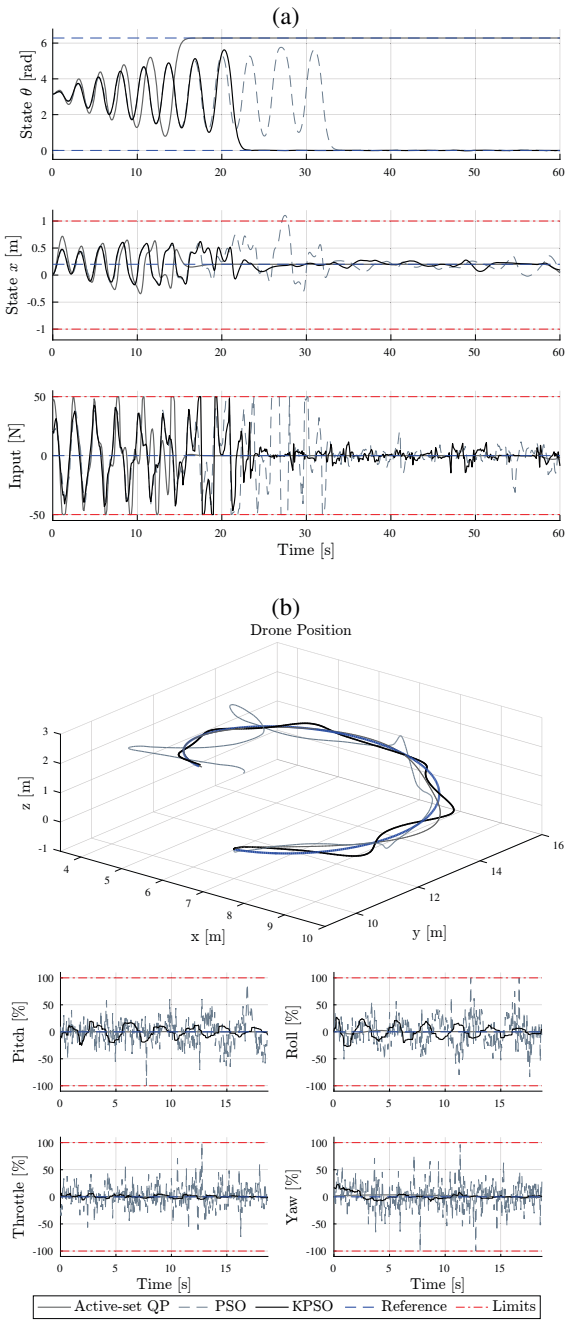

Fig. 5. NMPC simulation of (a) Inverted Pendulum, and (b) Quadrotor drone.

C. FPGA Accelerator Implementation Results

Two implementations of the NMPC algorithm were compared for the computational performance analysis. The first is a hardware/software co-design solution where the proposed NMPC accelerator runs in hardware. The second is a software-only solution in C++ running on the FPGA's ARM processor

TABLE III
EXECUTION TIME AND IMPLEMENTATION RESULT COMPARISON.

	Own HW		Own SW		[13]	[12]	[14]	[15]
System model	Quad. drone	Inv. pend	Quad. drone	Inv. pend	2-DoF SCARA	Skid drive	Skid drive	Inv.pend
Platform	Xilinx ZU3EG		ARM Cortex A53		Xilinx Kintex7	Altera Stratix III	Xilinx XC7Z020	Xilinx XC7Z045
Freq.	50MHz		1.3GHz		50MHz	50MHz	50MHz	100MHz
Perf. [ms]	29.67	11.16	251.51	58.5	7.2	7.56	0.083	2
S	5	10	5	10	60	30	-	-
N_h	25	100	25	100	2	5	5	10
States	12	4	12	4	2	3	4	6
Dim	4	1	4	1	2	2	2	2
Adaptable + systems	Yes		-		No	No	No	Limited
ROS comp.	Yes		-		No	No	No	No
LUT	53817	34384	-		-	46342	47923	100929
FF	60829	63564	-		-	42886	30059	67794
BRAM	39	84	-		-	-	22	75
DSP	145	11	-		-	192	78	670

on top of the Linux OS. The FPGA evaluation is done on an Ultra96-v2 development board that encloses a Zynq UltraScale+ MPSoC™ . This board is a relatively small device that was selected given that it is an excellent platform that can be used for low-power embedded applications and robotics. Similarly, this onboard evaluation leverages the ROS node described in Section V-D for both the SW and HW accelerated versions, subscribing to simulated sensor data generated by a software model of both the inverted pendulum and drone.

The performance results are summarized in Table III. It can be seen that the accelerator results are comparable to the execution on a Desktop PC and achieve a speed-up of approximately 5x and 8x for the inverted pendulum and quadrotor drone, respectively, with the ARM as a baseline and meets real-time constraints (See T_s in Table I).

Table III also describe other state-of-the-art implementations. The other deployments use different testbench systems, a 2DoF SCARA, and a skid-drive mobile robot. Given that the systems differ, the number of states and control variables are also mentioned for comparison purposes. The proposed solution shows a similar execution time for a more extensive N_h and S.

The Ultra96-v2 board used for this use case has an UltraScale+™ ZU3EG FPGA. The utilization report is summarized at the bottom for the device in Table III. It can be seen that even for a model as complex as the quadrotor drone, the architecture fits (approximately 44% of the target device, while 34% is used for the inverted pendulum). The results also show that the implementation has similar resource utilization to the state-of-the-art deployments, except for DSP, which is lower. On the other hand, the state-of-the-art implementations rely on big FPGAs with higher energy utilization, as demonstrated by benchmarks done on previous works [24]: $0.88mJ$, $0.90mJ$, and $2.9mJ$ for the Ultra96, Kintex7 and Stratix III respectively. These results indicate that this work is superior to the others, given that it can achieve a longer prediction horizon with a smaller device than can potentially drain less energy on runtime.

Finally, [12], [13] need to make more effort to make their architecture reusable for different applications and focus on the control of specific systems. On the other hand, this implementation brings a framework to add more non-linear systems to the accelerator that is also ROS compatible.

VII. CONCLUSIONS

This work presents a fast and parallel FPGA accelerator of an NMPC controller based on a PSO nonlinear solver. A novel PSO-NMPC algorithm is also proposed which can reach an improved control horizon with fewer iterations and particles than canonical PSO.

Two use cases were evaluated in this work, and the proposed hardware solution can compute each control effort deterministically (since the maximum iterations are fixed) with a low-end and low-cost MPSoC+FPGA device.

The controller accelerator is developed in HLS and is template based, enabling this solution to potentially be applied to other nonlinear systems, e.g., mobile robots or robot arms. It is also ROS compatible, which adds to its flexibility to be applied to other systems.

The architecture also shows superiority to other related works in three fronts:

i It can implement several systems. The work showed how this is possible while giving two use-case implementations, different from other works focusing only on one application.

ii It can comply with the real-time constraints of complex use-case models that require a long control horizon. Such is demonstrated with the quadrotor drone use case with 12 states, 4 dimensions, and a control horizon of 25. On the related compared works, the most prominent model use case was [14], and it had 6 states, 2 dimensions, and a control horizon of 10. The current work also shows lower

resource utilization (e.g., 47k LUT against 101k LUT in [14]).

iii It is more flexible as it is ROS-compatible and is ready to be deployed in real robotic systems.

Future works envision applying this architecture to more models to prove its flexibility further. It is also planned to improve the memory interconnect and management to reduce FPGA utilization. Similarly, including different solvers other than KPSO will be investigated to realize an even more flexible controller.

ACKNOWLEDGEMENTS

Funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy - EXC 2050/1 - Project ID 390696704 - Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden.

REFERENCES

[1] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A survey of linear model predictive control on fpgas," *IFAC-PapersOnLine*, 2018.

[2] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear MPC: bridging the gap via the real-time iteration," *International Journal of Control*, 2020.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*. IEEE.

[4] B. Khusainov, E. C. Kerrigan, and G. A. Constantinides, "Multi-objective Co-design for model predictive control with an FPGA," in *2016 European Control Conference (ECC)*, 2016.

[5] X. Lu, H. Chen, B. Gao, Z. Zhang, and W. Jin, "Data-Driven Predictive Gearshift Control for Dual-Clutch Transmissions and FPGA Implementation," *IEEE Transactions on Industrial Electronics*, 2015.

[6] L. Yuan and e. al., "Nonlinear MPC-based slip control for electric vehicles with vehicle safety constraints," *Mechatronics*, 2016.

[7] A. Raha, A. Chakrabarty, V. Raghunathan, and G. T. Buzzard, "Embedding approximate nonlinear model predictive control at ultrahigh speed and extremely low power," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 1092–1099, 2020.

[8] N. Guo, B. Lenzo, X. Zhang, Y. Zou, R. Zhai, and T. Zhang, "A real-time nonlinear model predictive controller for yaw motion optimization of distributed drive electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 4935–4946, 2020.

[9] F. Dong, X. Li, K. You, and S. Song, "Deep neural network based model predictive control for standoff tracking by a quadrotor uav," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 1810–1815.

[10] T. Weise, *Global Optimization Algorithms - Theory and Application*, 2nd ed. Self-Published, 2009.

[11] F. Xu, H. Chen, X. Gong, and Q. Mei, "Fast Nonlinear Model Predictive Control on FPGA Using Particle Swarm Optimization," *IEEE Transactions on Industrial Electronics*, 2016.

[12] F. Xu, Z. Guo, H. Chen, D. Ji, and T. Qu, "A custom parallel hardware architecture of nonlinear model-predictive control on fpga," *IEEE Transactions on Industrial Electronics*, 2022.

[13] S. P. Diwan and S. S. Deshpande, "Fast nonlinear model predictive controller using parallel pso based on divide and conquer approach," *International Journal of Control*, 2022.

[14] Y. Li, S. E. Li, X. Jia, S. Zeng, and Y. Wang, "FPGA accelerated model predictive control for autonomous driving," *Journal of Intelligent and Connected Vehicles*, vol. 5, no. 2, pp. 63–71, 2022.

[15] V. Patne, D. Ingole, and D. Sonawane, "Fpga implementation framework for low latency nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 7020–7025, 2020, 21st IFAC World Congress.

[16] D. Sevis, K. Senel, and Y. Denizhan, "A knowledge-supported improvement of the PSO method," *COMPEL - International Journal for Computation and Mathematics in Electrical and Electronic Eng.*, 2013.

[17] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of Modeling and Simulation*. Elsevier, 2019.

[18] D. G. Carta. (2005) Park-miller-carta pseudo-random number generator. [Online]. Available: https://www.firstpr.com.au/dsp/rand31

[19] A. Podlubne, J. Mey, R. Schöne, U. Aßmann, and D. Göhringer, "Model-based approach for automatic generation of hardware architectures for robotics," *IEEE Access*, 2021.

[20] K. Åström and K. Furuta, "Swinging up a pendulum by energy control," *Automatica*, 2000.

[21] T. Luukkonen, "Modelling and control of quadcopter," Master's thesis, Alto University, 2011.

[22] A. Engelbrecht, "Computational intelligence: An introduction." Wiley, 2007, ch. 16.

[23] P. E. Gill, W. Murray, and M. H. Wright, in *Practical Optimization*. Academic Press, 1981, ch. 5.

[24] G. Dinelli, G. Meoni, E. Rapuano, G. Benelli, and L. Fanucci, "An FPGA-based hardware accelerator for CNNs using on-chip memories only," *International Journal of Reconfigurable Computing*, 2019.